

Memoria PI5

Yeray Rincón Cardoso

Ejercicio 1

Datos

```
package Datos;
```

```
import java.util.ArrayList;
```

```
import java.util.HashSet;
```

```
import java.util.LinkedList;
```

```
import java.util.List;
```

```
import java.util.Set;
```

```
import us.lsi.common.Files2;
```

```
public class DatosAgricultor {
```

```
    private static List<Integer> huertos;
```

```
    private static List<Integer> variedades;
```

```
    private static Set<Set<Integer>> incompatibility;
```

```
    public record Huerto(String nombre, Integer metrosDisponibles) {
```

```
        public static Huerto of(String nombre, Integer metrosDisponibles) {
```

```
            return new Huerto(nombre, metrosDisponibles);
```

```
        }
```

```
        public static Huerto parseaHuerto(String cadena) {
```

```

        String[] partes = cadena.split(":");

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

        String[] p = partes[1].trim().split("=");
        String metrosDisponibles = p[1].trim().replace(";",
        "").trim());

        return Huerto.of(nombre,
        Integer.valueOf(metrosDisponibles));
    }
}

```

```

    public record Variedad(String nombre, Integer
    metrosRequeridos, List<String> comp) {

```

```

        public static Variedad of(String nombre, Integer
        metrosRequeridos, List<String> comp) {
            return new Variedad(nombre, metrosRequeridos,
            comp);
        }

```

```

    public static Variedad parseaVariedad(String cadena) {
        String[] partes = cadena.split("->");
        String nombre = partes[0].trim();
        String[] p = partes[1].split(";");
        String[] p2 = p[0].split("=");
        String metrosRequeridos = p2[1].trim();
        String cadenaComponentes = p[1].trim();

```

```

        List<String> comp =
listaVariedades(cadenaComponentes);

        return Variedad.of(nombre,
Integer.valueOf(metrosRequeridos), comp);
    }

```

```

private static List<String> listaVariedades(String cadena)
{
    List<String> res = new ArrayList<>();
    String[] partes = cadena.split("=");

    String[] p = partes[1].trim().split(",");
    for (String cadVariedad : p) {
        String v = cadVariedad;
        res.add(v);
    }
    return res;
}

}

```

```

public static void iniDatos(String file) {
    List<String> listaHuertos = Files2.linesFromFile(file);
    huertos = new LinkedList<>();
    variedades = new LinkedList<>();
    incompatibility = new HashSet<>();
    for (int i = 0; i < listaHuertos.size(); i++) {
        String l = listaHuertos.get(i);

```

```

        if (l.contains(":")) {
            String[] v = l.replace(";", "").split("=");
            huertos.add(Integer.parseInt(v[1]));
        } else if (l.contains("->")) {
            String[] v = l.split(";");
            String[] v2 = v[0].split("=");
            variedades.add(Integer.parseInt(v2[1]));
            v2 = v[1].split("=");
            String aux = v2[1].replaceAll("V", "");
            ;
            Set<Set<Integer>> incom = new
HashSet<>();
            for (String s : aux.split(",")) {
                Set<Integer> pareja = new
HashSet<Integer>();
                pareja.add(variedades.size() - 1);
                pareja.add(Integer.parseInt(s));
                incom.add(pareja);
            }
            incompatibility.addAll(incom);
        }
    }
    toConsole();
}

```

```

public static void main(String[] args) {
    for (int i = 0; i < 3; i++) {
        System.out.println("Fichero de entrada: " + (i + 1));
    }
}

```

```
        iniDatos("Ficheros/Ejercicio1DatosEntrada" + (i + 1)
+ ".txt");
```

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

```
    }
```

```
}
```

```
public static List<Integer> getHuertos() {
```

```
    return huertos;
```

```
}
```

```
public static Integer getHuertoI(Integer i) {
```

```
    return huertos.get(i);
```

```
}
```

```
public static Integer getVariedadI(Integer i) {
```

```
    return variedades.get(i);
```

```
}
```

```
public static Integer getNumHuertos() {
```

```
    return huertos.size();
```

```
}
```

```
public static List<Integer> getVariedades() {
```

```
    return variedades;
```

```
}
```

```

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

    public static Integer getIncompatibilidad(Integer i, Integer k) {
        Set<Integer> conj = new HashSet<>();
        conj.add(i);
        conj.add(k);
        return incompatibility.contains(conj) ? 1 : 0;
    }

    public static Set<Set<Integer>> getVarCompatibilidad() {
        return incompatibility;
    }

    private static void toConsole() {
        System.out.println("Variedades disponibles: " +
            variedades + "\nMetros cuadrados disponibles por huerto: "
            + huertos + "NumHuertos: " +
            getNumHuertos());
    }
}

```

Solucion

```
package Soluciones;
```

```
import java.util.HashMap;
```

```
import java.util.List;
import java.util.Map;
```

```
import org.jgrapht.GraphPath;
```

```
import Datos.DatosAgricultor;
import Ejercicio1.HuertosEdge;
import Ejercicio1.HuertosVertex;
import us.lsi.common.List2;
```

```
public record SolucionAgricultor(Double weight, Map<Integer,
List<Integer>> map) implements Comparable<SolucionAgricultor>{
```

```
    public static SolucionAgricultor of(GraphPath<HuertosVertex,
HuertosEdge> camino) {
```

```
        List<Integer> list = camino.getEdgeList().stream().map(x ->
x.action()).toList();
```

```
        return SolucionAgricultor.of(list);
```

```
    }
```

```
    public static SolucionAgricultor of(List<Integer> value) {
```

```
        Map<Integer, List<Integer>> map = new HashMap<>();
```

```
        Double weight = (double) value.stream().filter(x -> x !=
DatosAgricultor.getNumHuertos()).count();
```

```
        Integer cont = 0;
```

```
        for (Integer i: value) {
```

```
            if (i < DatosAgricultor.getNumHuertos()) {
```

```
                if (map.containsKey(i)) {
```

```
                    map.get(i).add(cont);
```

```

        } else {
            List<Integer> listaReparto =
List2.empty();

            listaReparto.add(cont);
            map.put(i, listaReparto);
        }
    }
    cont++;
}
return new SolucionAgricultor(weight, map);
}

```

```

@Override
public int compareTo(SolucionAgricultor o) {
    // TODO Auto-generated method stub
    return this.weight().compareTo(o.weight());
}
}

```

Edge

```
package Ejercicio1;
```

```
import Datos.DatosAgricultor;
```

```
import us.lsi.graphs.virtual.SimpleEdgeAction;
```

```

public record HuertosEdge(HuertosVertex source, HuertosVertex
target, Integer action, Double weight) implements
SimpleEdgeAction<HuertosVertex, Integer> {

```



```

    public static HuertosEdge of(HuertosVertex source, HuertosVertex
target, Integer action) {
        Double peso = 0.;
        if (action != DatosAgricultor.getNumHuertos()) {
            peso=1.;
        }
        return new HuertosEdge(source, target, action, peso);
    }
}

```

Vertex

```
package Ejercicio1;
```

```

import java.util.List;
import java.util.Set;
import java.util.function.Predicate;

```

```

import Datos.DatosAgricultor;
import us.lsi.common.IntegerSet;
import us.lsi.common.List2;
import us.lsi.graphs.virtual.VirtualVertex;

```

```

public record HuertosVertex(Integer index, List<IntegerSet> reparto,
List<Integer> metrosDisponibles) implements
VirtualVertex<HuertosVertex, HuertosEdge, Integer> {

```

```

    public static HuertosVertex initial() {
        List<IntegerSet> reparto = List2.empty();
        List<Integer> metrosDisp = List2.empty();
    }
}

```

```

        for (int i = 0; i < DatosAgricultor.getNumHuertos(); i++) {

metrosDisp.add(DatosAgricultor.getHuertos().get(i));
            reparto.add(IntegerSet.empty());
        }
        return new HuertosVertex(0, reparto, metrosDisp);
    }

    public List<Integer> actions() {
        List<Integer> actions =
List2.of(DatosAgricultor.getNumHuertos());
        if (index() >= DatosAgricultor.getNumVariedades()) {
            return List2.empty();
        } else {
            for (int i = 0; i < DatosAgricultor.getNumHuertos();
i++) {

                Set<Integer> huerto = reparto.get(i);

                Integer mRest = metrosDisponibles.get(i) -
DatosAgricultor.getVariedadI(index);

                if(mRest >= 0 &&
huerto.stream().noneMatch(x ->
DatosAgricultor.getIncompatibilidad(x, index) == 1)) {
                    actions.add(i);
                }
            }
            return actions;
        }
    }
}

```

```

    public HuertosVertex neighbor(Integer a) {
        List<IntegerSet> reparto = List2.copy(this.reparto);
        List<Integer> disp = List2.copy(this.metrosDisponibles);

        if(a != DatosAgricultor.getNumHuertos()) {
            Integer diff = metrosDisponibles().get(a) -
            DatosAgricultor.getVariedadI(this.index);
            metrosDisponibles.set(a, diff);
            IntegerSet copia = IntegerSet.copy(this.reparto.get(a));
            copia.add(index());
            reparto.set(a, copia);
        }
        return new HuertosVertex(this.index + 1, reparto, disp);
    }

```

```

@Override
    public HuertosEdge edge(Integer a) {
        return HuertosEdge.of(this, neighbor(a), a);
    }

```

```

    public static Predicate<HuertosVertex> goal() {
        return x -> x.index() == DatosAgricultor.getNumVariedades();
    }

```

```

    public static Predicate<HuertosVertex> goalHasSolution() {
        return x -> true;
    }

```

```

        public String toGraph() {
            return String.format("%d, %s, %s", this.index, this.reparto,
this.metrosDisponibles);
        }

```

```

        public static void main(String[] args) {

```

```

            DatosAgricultor.iniDatos("Ficheros/Ejercicio1DatosEntrada1.txt
");
            System.out.println(initial());
        }

```

```

    }

```

Heuristic

```

package Ejercicio1;

```

```

import java.util.Set;

```

```

import java.util.function.Predicate;

```

```

import Datos.DatosAgricultor;

```

```

public class HuertosHeuristic {

```

```

        public static Double heuristic(HuertosVertex v1,
Predicate<HuertosVertex> goal, HuertosVertex v2) {
            Double res = 0.;

```

```

        for (int i = v1.index(); i <
DatosAgricultor.getNumVariedades(); i++) {
            for (int j = 0; j <
DatosAgricultor.getNumHuertos(); j++) {
                Set<Integer> huerto =
v1.reparto().get(j);
                Integer metrosRestantes =
v1.metrosDisponibles().get(j) - DatosAgricultor.getVariedadI(i);
                Integer var = i;
                if (metrosRestantes >= 0
                    &&
huerto.stream().noneMatch(x ->
DatosAgricultor.getIncompatibilidad(var, x) == 1)) {
                    res += 1.;
                }
            }
        }
    }
    return res;
}
}

```

PD

```
package Ejercicio1_Manual;
```

```
import java.util.Comparator;
```

```
import java.util.List;
```

```
import java.util.Map;
```

```
import Datos.DatosAgricultor;
```

```
import Soluciones.SolucionAgricultor;
```

```

import us.lsi.common.List2;
import us.lsi.common.Map2;

public class HuertosPD {

    public static record Spm(Integer a, Integer weight) implements
Comparable<Spm> {
        public static Spm of(Integer a, Integer weight) {
            return new Spm(a, weight);
        }

        @Override
        public int compareTo(Spm sp) {
            return this.weight.compareTo(sp.weight);
        }
    }

    public static Map<HuertosProblem, Spm> memory;
    public static Integer mejorValor = Integer.MIN_VALUE;

    public static SolucionAgricultor search() {
        memory = Map2.empty();
        mejorValor = Integer.MIN_VALUE;    // Estamos
maximizando

        pdr_search(HuertosProblem.initial(), 0);
        return getSolucion();
    }
}

```

```

        private static Double acotar(Integer acum, HuertosProblem
origen, Integer accion) {
            Integer weight = 0;
            if (accion != DatosAgricultor.getNumHuertos())
                weight = 1;
            if (accion != DatosAgricultor.getNumHuertos()) {
                weight = 1;
            }
            return acum + weight +
origen.neighbor(accion).heuristic();
        }

```

```

public static SolucionAgricultor getSolucion() {
    List<Integer> acciones = List2.empty();
    HuertosProblem prob = HuertosProblem.initial();
    Spm spm = memory.get(prob);
    while (spm != null && spm.a != null) {
        HuertosProblem old = prob;
        acciones.add(spm.a);
        prob = old.neighbor(spm.a);
        spm = memory.get(prob);
    }
    return SolucionAgricultor.of(acciones);
}

```

```

private static Spm pdr_search(HuertosProblem problema,
Integer acumulado) {

```

```

Boolean esGoal = HuertosProblem.goal().test(problema);
if (memory.containsKey(problema)) {
    return memory.get(problema);
}
else if (esGoal) {
    Spm solucion_cb = Spm.of(null, 0);
    memory.put(problema, solucion_cb);
    if (acumulado > mejorValor) {
        mejorValor = acumulado;
    }
    return solucion_cb;
}
// 3. En cualquier otro caso llamada recursiva
else {
    List<Spm> solucionesPosibles = List2.empty();
    for (Integer action : problema.actions()) {
        Double estimacion = acotar(acumulado,
problema, action);
        if (estimacion < mejorValor)
            continue;
        HuertosProblem vecino =
problema.neighbor(action);
        Integer weight = 0;
        if (action != DatosAgricultor.getNumHuertos())
            weight = 1;
        Spm solucionVecino = pdr_search(vecino,
acumulado + weight);
        if (solucionVecino != null) {

```



```
        Spm solucionBuena = Spm.of(action,
solucionVecino.weight() + weight);
```

```
        solucionesPosibles.add(solucionBuena);
```

```
    }
```

```
    }
```

```
    // Busco la maxima porque estoy maximizando
```

```
        Spm mejorSolucion =
solucionesPosibles.stream().max(Comparator.naturalOrder()).orElse
(null);
```

```
        if (mejorSolucion != null) {
```

```
            memory.put(problema, mejorSolucion);
```

```
            return mejorSolucion;
```

```
        }
```

```
    }
```

```
    return null;
```

```
}
```

```
}
```

Problem

```
package Ejercicio1_Manual;
```

```
import java.util.List;
```

```
import java.util.Set;
```

```
import java.util.function.Predicate;
```

```
import Datos.DatosAgricultor;
```

```
import us.lsi.common.IntegerSet;
```

```
import us.lsi.common.List2;
```

```
public record HuertosProblem(Integer indice, List<IntegerSet>
reparto, List<Integer> metrosDisponibles) {
```

```
    public static HuertosProblem initial() {
        List<Integer> metros = List2.empty();
        List<IntegerSet> reparto = List2.empty();
        for(int i = 0; i < DatosAgricultor.getNumHuertos(); i++) {
            metros.add(DatosAgricultor.getHuertoI(i));
            reparto.add(IntegerSet.empty());
        }
        return new HuertosProblem(0, reparto, metros);
    }
```

```
    public static Predicate<HuertosProblem> goal() {
        return obj -> obj.indice() ==
DatosAgricultor.getNumVariedades();
    }
```

```
    public static Predicate<HuertosProblem> goalHasSolution() {
        return obj -> true;
    }
```

```
    public List<Integer> actions() {
        List<Integer> alternativas =
List2.of(DatosAgricultor.getNumHuertos());
        if(indice() >= DatosAgricultor.getNumVariedades()) {
            return List2.empty();
        } else {
```

```

        for(int i = 0; i<DatosAgricultor.getNumHuertos(); i++) {
            Set<Integer> huerto = reparto.get(i);
            Integer metrosRestantes = metrosDisponibles.get(i) -
DatosAgricultor.getVariedadI(indice());
            if(metrosRestantes >= 0 && huerto.stream().noneMatch(x -
> DatosAgricultor.getIncompatibilidad(x, indice()) == 1)) {
                alternativas.add(i);
            }
        }
        return alternativas;
    }
}

```

```

public HuertosProblem neighbor(Integer a) {
    List<IntegerSet> repartoSiguiente = List2.copy(this.reparto);
    List<Integer> metrosDisponiblesSiguientes =
List2.copy((this.metrosDisponibles));
    if(a < DatosAgricultor.getNumHuertos()) {
        Integer diferencia = metrosDisponibles().get(a) -
DatosAgricultor.getVariedadI(this.indice);
        metrosDisponiblesSiguientes.set(a, diferencia);
        IntegerSet copiaHuerto = IntegerSet.copy(reparto.get(a));
        copiaHuerto.add(indice());
        repartoSiguiente.set(a, copiaHuerto);
    }
    return new HuertosProblem(this.indice + 1, repartoSiguiente,
metrosDisponiblesSiguientes);
}

```

```

public Double heuristic() {
    return auxiliar(this, DatosAgricultor.getNumVariedades());
}

private static Double auxiliar(HuertosProblem vertice, Integer
ultimoIndice){
    Double numVar = 0.;
    for(int i = vertice.indice(); i < ultimoIndice; i++) {
        for(int j = 0; j<DatosAgricultor.getNumHuertos(); j++) {
            Set<Integer> huerto = vertice.reparto().get(j);
            Integer metrosRestantes =
vertice.metrosDisponibles().get(j) - DatosAgricultor.getVariedadI(i);
            Integer variedad = i;
            if(metrosRestantes >= 0 && huerto.stream().noneMatch(x -
> DatosAgricultor.getIncompatibilidad(variedad, x) == 1)) {
                numVar += 1;
            }
        }
    }
    return numVar;
}

public static void main(String[] args) {

DatosAgricultor.iniDatos("ficheros/Ejercicio1DatosEntrada3.txt");
    System.out.println(initial());
}
}

```

Test A*

```
package Tests;
```

```
import java.util.List;
```

```
import java.util.Locale;
```

```
import org.jgrapht.GraphPath;
```

```
import Datos.DatosAgricultor;
```

```
import Ejercicio1.HuertosEdge;
```

```
import Ejercicio1.HuertosHeuristic;
```

```
import Ejercicio1.HuertosVertex;
```

```
import Soluciones.SolucionAgricultor;
```

```
import us.lsi.colors.GraphColors;
```

```
import us.lsi.colors.GraphColors.Color;
```

```
import us.lsi.graphs.alg.AStar;
```

```
import us.lsi.graphs.virtual.EDGraph;
```

```
import us.lsi.graphs.virtual.EDGraph.Type;
```

```
import us.lsi.path.EDGraphPath.PathType;
```

```
public class Ejercicio1Test {
```

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

```
        Locale.setDefault(Locale.of("en", "US"));
```

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

```
DatosAgricultor.iniDatos("Ficheros/Ejercicio1DatosEntrada" + i  
+ ".txt");
```

```
System.out.println("\n\n>\tResultados para el test " + i + "\n");
```

```
HuertosVertex start = HuertosVertex.initial();
```

```
EGraph<HuertosVertex, HuertosEdge> grafo =  
    EGraph.virtual(start,          HuertosVertex.goal(),  
    PathType.Sum, Type.Max)  
    .edgeWeight(x -> x.weight())
```

```
.goalHasSolution(HuertosVertex.goalHasSolution())  
    .heuristic(HuertosHeuristic::heuristic)  
    .build();
```

```
System.out.println("\n\n#### Ejercicio 1 Algoritmo A* ####");
```

```
AStar<HuertosVertex, HuertosEdge, ?> astar =  
    AStar.of(grafo);
```

```
GraphPath<HuertosVertex, HuertosEdge> camino =  
    astar.search().get();
```

```
List<Integer> camino_as =  
    camino.getEdgeList().stream()  
    .map(x -> x.action())  
    .toList();
```

```
SolucionAgricultor sol = SolucionAgricultor.of(camino_as);  
System.out.println(sol);
```

```
GraphColors.toDot(astar.outGraph(),  
    "Grafos_Generados/Ejercicio1/Ejercicio1Auto" + i +  
".gv",  
    v -> v.toGraph(),  
    e -> e.action().toString() + ", " + e.weight().toString(),  
    v -> GraphColors.colorIf(Color.green,  
HuertosVertex.goal().test(v)),  
    e -> GraphColors.colorIf(Color.green,  
(camino.getEdgeList().contains(e))));  
}  
}  
  
}
```

```

Variedades disponibles: [2, 4, 3, 1, 6]
Metros cuadrados disponibles por huerto: [4, 6]NumHuertos: 2

>      Resultados para el test 1

#### Ejercicio 1 Algoritmo A* ####
SolucionAgricultor[weight=4.0, map={0=[0, 1], 1=[2, 3]}}
Variedades disponibles: [4, 2, 3, 1, 6, 2]
Metros cuadrados disponibles por huerto: [3, 4, 5]NumHuertos: 3

>      Resultados para el test 2

#### Ejercicio 1 Algoritmo A* ####
SolucionAgricultor[weight=5.0, map={0=[1, 2], 1=[0, 5], 2=[3]}}
Variedades disponibles: [6, 4, 3, 1, 6, 5, 2, 5, 2]
Metros cuadrados disponibles por huerto: [10, 5, 2, 8]NumHuertos: 4

>      Resultados para el test 3

#### Ejercicio 1 Algoritmo A* ####
SolucionAgricultor[weight=8.0, map={0=[0, 5, 7], 1=[1, 3], 2=[8], 3=[2, 4]}}

```

Test BT

```
package Tests;
```

```
import java.util.List;
```

```
import java.util.Locale;
```

```
import java.util.Optional;
```

```
import org.jgrapht.GraphPath;
```

```
import Datos.DatosAgricultor;
```

```
import Ejercicio1.HuertosEdge;
```

```
import Ejercicio1.HuertosHeuristic;
```

```
import Ejercicio1.HuertosVertex;
```

```
import Soluciones.SolucionAgricultor;
```



```

import us.lsi.colors.GraphColors;
import us.lsi.colors.GraphColors.Color;
import us.lsi.graphs.alg.BT;
import us.lsi.graphs.alg.GreedyOnGraph;
import us.lsi.graphs.virtual.EDGraph;
import us.lsi.graphs.virtual.EDGraph.Type;
import us.lsi.path.EDGraphPath.PathType;

public class Ejercicio1TestBT {

    public static void main(String[] args) {

        Locale.setDefault(Locale.of("en", "US"));

        for (int i = 1; i <= 3; i++) {
            DatosAgricultor.iniDatos("Ficheros/Ejercicio1DatosEntrada"
+ i + ".txt");
            System.out.println("\n\n>\tResultados para el test " + i + "\n");

            HuertosVertex start = HuertosVertex.initial();

            EDGraph<HuertosVertex, HuertosEdge> grafo =
                EDGraph.virtual(start,                HuertosVertex.goal(),
PathType.Sum, Type.Max)
                .edgeWeight(x -> x.weight())
                .goalHasSolution(HuertosVertex.goalHasSolution())
                .heuristic(HuertosHeuristic::heuristic)
                .build();

```

```
System.out.println("\n\n#### Ejercicio 1 Algoritmo BT ####");
```

```
Boolean conVoraz = false;
```

```
SolucionAgricultor sv = null;
```

```
Optional<GraphPath<HuertosVertex, HuertosEdge>> gp =  
Optional.empty();
```

```
if (conVoraz) {
```

```
    GreedyOnGraph<HuertosVertex, HuertosEdge> ga =  
    GreedyOnGraph.of(grafo);
```

```
    gp = ga.search();
```

```
    if (gp.isPresent()) sv = SolucionAgricultor.of(gp.get());
```

```
    System.out.println("Sv = "+sv);
```

```
}
```

```
BT<HuertosVertex, HuertosEdge, SolucionAgricultor> bta =  
null;
```

```
if(gp.isPresent())
```

```
    bta = BT.of(grafo, SolucionAgricultor::of,  
gp.get().getWeight(), gp.get(), true);
```

```
else
```

```
    bta = BT.of(grafo, null, null, null, true);
```

```
bta.search();
```

```
sv = SolucionAgricultor.of(bta.optimalPath().orElse(null));
```

```

List<HuertosEdge> le = bta.optimalPath().get().getEdgeList();

System.out.println("Sol opt = "+sv);

var outGraph = bta.outGraph();

if(outGraph!=null)
    GraphColors.toDot(bta.outGraph(),
        "Grafos_Generados/Ejercicio1/Ejercicio1BT"+i+".gv",
        v -> v.toGraph(),
        e -> e.action().toString(),
        v -> GraphColors.colorIf(Color.red,
HuertosVertex.goal().test(v)),
        e -> GraphColors.colorIf(Color.red, le.contains(e)));

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

```

```

Variedades disponibles: [2, 4, 3, 1, 6]
Metros cuadrados disponibles por huerto: [4, 6]NumHuertos: 2

> Resultados para el test 1

#### Ejercicio 1 Algoritmo BT ####
Sol opt = SolucionAgricultor[weight=4.0, map={0=[0, 1], 1=[2, 3]}]

Variedades disponibles: [4, 2, 3, 1, 6, 2]
Metros cuadrados disponibles por huerto: [3, 4, 5]NumHuertos: 3

> Resultados para el test 2

#### Ejercicio 1 Algoritmo BT ####
Sol opt = SolucionAgricultor[weight=5.0, map={0=[1, 2], 1=[0, 5], 2=[3]}]

Variedades disponibles: [6, 4, 3, 1, 6, 5, 2, 5, 2]
Metros cuadrados disponibles por huerto: [10, 5, 2, 8]NumHuertos: 4

> Resultados para el test 3

#### Ejercicio 1 Algoritmo BT ####
Sol opt = SolucionAgricultor[weight=8.0, map={0=[0, 5, 7], 1=[3, 8], 2=[6], 3=[2, 4]}]

```

Test PDR

```
package Tests;
```

```
import java.util.Locale;
```

```
import java.util.Optional;
```

```
import java.util.function.Predicate;
```

```
import org.jgrapht.GraphPath;
```

```
import Datos.DatosAgricultor;
```

```
import Ejercicio1.HuertosEdge;
```

```
import Ejercicio1.HuertosHeuristic;
```

```
import Ejercicio1.HuertosVertex;
```

```
import Soluciones.SolucionAgricultor;
```

```
import us.lsi.colors.GraphColors;
```

```

import us.lsi.colors.GraphColors.Color;
import us.lsi.graphs.alg.GreedyOnGraph;
import us.lsi.graphs.alg.PDR;
import us.lsi.graphs.virtual.EDGraph;
import us.lsi.graphs.virtual.EDGraph.Type;
import us.lsi.path.EDGraphPath.PathType;

public class Ejercicio1TestPDR {
    public static void main(String[] args) {
        Locale.setDefault(Locale.of("en", "US"));

        for (Integer id_fichero = 1; id_fichero <= 3; id_fichero++) {

            DatosAgricultor.iniDatos("Ficheros/Ejercicio1DatosEntrada"+id_fiche
            ro+".txt");

            HuertosVertex vInicial = HuertosVertex.initial();
            Predicate<HuertosVertex> goal = HuertosVertex.goal();

            EDGraph<HuertosVertex, HuertosEdge> graph =
                EDGraph.virtual(vInicial, goal, PathType.Sum, Type.Max)
                    .goalHasSolution(HuertosVertex.goalHasSolution())
                    .heuristic(HuertosHeuristic::heuristic)
                    .build();

            GreedyOnGraph<HuertosVertex, HuertosEdge> alg_voraz =
            GreedyOnGraph.of(graph);

            GraphPath<HuertosVertex, HuertosEdge> path =
            alg_voraz.path();

```

```
path = alg_voraz.isSolution(path)? path: null;
```

```
PDR<HuertosVertex,HuertosEdge,SolucionAgricultor>  
alg_pdr = path==null?
```

```
    PDR.of(graph):
```

```
    PDR.of(graph, null, path.getWeight(), path, true);
```

```
Optional<GraphPath<HuertosVertex, HuertosEdge>> gp =  
alg_pdr.search();
```

```
var res = alg_pdr.search().orElse(null);
```

```
var outGraph = alg_pdr.outGraph();
```

```
if(outGraph!=null) {
```

```
    GraphColors.toDot(alg_pdr.outGraph,
```

```
"Grafos_Generados/Ejercicio1/Ejercicio1PDR"+id_fichero+".gv",
```

```
        v -> v.toGraph(),
```

```
        e -> e.action().toString(),
```

```
        v -> GraphColors.colorIf(Color.red,  
HuertosVertex.goal().test(v)),
```

```
        e -> GraphColors.colorIf(Color.red,  
gp.isPresent()?gp.get().getEdgeList().contains(e):false));
```

```
    }
```

```
if(res!=null)
```

```
    System.out.println("Solucion PDR: " +  
SolucionAgricultor.of(res) + "\n");
```

```
else
```

```
    System.out.println("PDR no obtuvo solucion\n");
```

```

    }
}
}

Problems Javadoc Declaration Console X
<terminated> Ejercicio1TestPDR [Java Application] C:\Program Files\Java\jdk-20\bin\javaw.exe (12 may 2024 10:13:29 - 10:13:30) [pid: 22836]
Variedades disponibles: [2, 4, 3, 1, 6]
Metros cuadrados disponibles por huerto: [4, 6]NumHuertos: 2
Solucion PDR: SolucionAgricultor[weight=0.0, map={}]

Variedades disponibles: [4, 2, 3, 1, 6, 2]
Metros cuadrados disponibles por huerto: [3, 4, 5]NumHuertos: 3
Solucion PDR: SolucionAgricultor[weight=3.0, map={0=[1, 2], 2=[3]}]

Variedades disponibles: [6, 4, 3, 1, 6, 5, 2, 5, 2]
Metros cuadrados disponibles por huerto: [10, 5, 2, 8]NumHuertos: 4
Solucion PDR: SolucionAgricultor[weight=6.0, map={0=[1, 3], 1=[5, 7], 2=[6], 3=[8]}]

```

Test Manual

```

package Tests_Manuales;

import java.util.List;

import Datos.DatosAgricultor;
import Ejercicio1_Manual.HuertosPD;
import us.lsi.common.String2;

public class Ejercicio1Test_Manul {
    public static void main(String[] args) {
        List.of(1,2,3).forEach(num_test -> {

            System.out.println("*****
FICHERO " + num_test + "*****");

            DatosAgricultor.iniDatos("Ficheros/Ejercicio1DatosEntrada"+n
um_test+".txt");

            String2.toConsole("Solucion del ejercicio 1 usando
programacion dinamica manual para el fichero %s: %s\n", num_test,
HuertosPD.search());

```

```
});  
  
}  
  
}
```

```
Problems • Javadoc • Declaration • Console X  
<terminated> Ejercicio1Test Manul [Java Application] C:\Program Files\Java\jdk-20\bin\java.exe (12 may 2024 10:13:46 - 10:13:47) [pid: 23968]  
*****FICHERO 1*****  
Variedades disponibles: [2, 4, 3, 1, 6]  
Metros cuadrados disponibles por huerto: [4, 6]NumHuertos: 2  
Solucion del ejercicio 1 usando programacion dinamica manual para el fichero 1: SolucionAgricultor[weight=4.0, map={0=[2, 3], 1=[0, 1]}]  
*****FICHERO 2*****  
Variedades disponibles: [4, 2, 3, 1, 6, 2]  
Metros cuadrados disponibles por huerto: [3, 4, 5]NumHuertos: 3  
Solucion del ejercicio 1 usando programacion dinamica manual para el fichero 2: SolucionAgricultor[weight=5.0, map={0=[3, 5], 1=[0], 2=[1, 2]}]  
*****FICHERO 3*****  
Variedades disponibles: [6, 4, 3, 1, 6, 5, 2, 5, 2]  
Metros cuadrados disponibles por huerto: [10, 5, 2, 8]NumHuertos: 4  
Solucion del ejercicio 1 usando programacion dinamica manual para el fichero 3: SolucionAgricultor[weight=7.0, map={0=[5, 7], 1=[1, 3], 2=[6], 3=[0, 8]}]
```


Ejercicio 2

Datos

```
package Datos;
```

```
import java.util.List;
```

```
import java.util.stream.Collectors;
```

```
import us.lsi.common.Files2;
```

```
import us.lsi.common.String2;
```

```
public class DatosProductos {
```

```
    public static Integer presupuesto;
```

```
    public static List<Producto> productos;
```

```
    public record Producto(Integer id, Integer precio, Integer  
categoria, Integer valoracion) {
```

```
        public static Producto create(String s) {
```

```
            String[] productos = s.split(":");
```

```
            Integer id = Integer.valueOf(productos[0].trim());
```

```
            Integer                precio                =  
Integer.valueOf(productos[1].trim());
```

```
            Integer                categoria                =  
Integer.valueOf(productos[2].trim());
```

```
            Integer                valoracion                =  
Integer.valueOf(productos[3].trim());
```

```
            return new Producto(id, precio, categoria,  
valoracion);
```

```
        }
```

```

        public String toString() {
            return "Producto: " + id + ", Precio: " + precio + ",
Categoria: " + categoria + ", Valoracion: "
                + valoracion;
        }
    }

```

```

    public static Integer getPresupuesto() {
        return presupuesto;
    }

```

```

    public static List<Producto> getProductos() {
        return productos;
    }

```

```

    public static Integer getNumProductos() {
        return productos.size();
    }

```

```

    public static Integer getM() {
        return (int) productos.stream().map(p ->
p.categoria()).distinct().count();
    }

```

```

    public static Integer getCategoria(Integer i, Integer j) {
        return productos.get(i).categoria().equals(j) ? 1 : 0;
    }

```

```

public static Integer getCategoria(Integer i) {
    return productos.get(i).categoria();
}

```

```

public static Integer getPrecioProducto(Integer i) {
    return productos.get(i).precio();
}

```

```

public static Integer getValoracionProducto(Integer i) {
    return productos.get(i).valoracion();
}

```

```

public static Integer getNumCategorias() {
    return (int) productos.stream().map(p
p.categoria).count();
}

```

```

public static void iniDatos(String fichero) {
    presupuesto = Files2.streamFromFile(fichero).filter(x ->
x.contains("Presupuesto"))

```

```

    .map(DatosProductos::parseaPresupuesto).findFirst().get();
    productos =
Files2.streamFromFile(fichero).skip(2).map(Producto::create).toList(
);

```

```

    String s1 = "Presupuesto: " + presupuesto.toString();
    String2.toConsole("%s\n%s", s1, String2.linea());
    String s2 =
productos.stream().map(Producto::toString).collect(Collectors.joining
("\n"));

```

```

        String2.toConsole("%s\n%s", s2, String2.linea());
    }

    private static Integer parseaPresupuesto(String s) {
        String[] presupuesto = s.split(" = ");
        return Integer.valueOf(presupuesto[1].trim());
    }

    public static void main(String[] args) {
        iniDatos("ficheros/Ejercicio2DatosEntrada2.txt");
    }
}

```

Solucion

```
package Soluciones;
```

```

import java.util.ArrayList;
import java.util.List;
import java.util.stream.Collectors;

```

```
import org.jgrapht.GraphPath;
```

```

import Datos.DatosProductos;
import Ejercicio2.ProductosEdge;
import Ejercicio2.ProductosVertex;

```

```

public record SolucionProductos(Double precioTotal, List<Integer>
productos, Integer acumValoracion) implements
Comparable<SolucionProductos> {

```

```

    public static SolucionProductos of(GraphPath<ProductosVertex,
ProductosEdge> path) {
        List<Integer> la = path.getEdgeList().stream().map(e ->
e.action()).toList();
        return SolucionProductos.of(la);
    }

```

```

public static SolucionProductos of(List<Integer> value) {
    List<Integer> pS = new ArrayList<>();
    Double pT = 0.;
    Integer aV = 0;
    for (int i = 0; i < value.size(); i++) {
        if (value.get(i) == 1) {
            pS.add(i);
            pT += DatosProductos.getPrecioProducto(i);
            aV += DatosProductos.getValoracionProducto(i) - 3;
        }
    }
    return new SolucionProductos(pT, pS, aV);
}

```

```

@Override
public int compareTo(SolucionProductos o) {
    return this.precioTotal().compareTo(o.precioTotal());
}

```

```

@Override

```

```

    public String toString() {
        String s = productos.stream().map(p -> "Producto" + p)
            .collect(Collectors.joining(", ", "{", "} \n Precio Total:" +
                precioTotal));
        return s;
    }
}

```

Edge

```

package Ejercicio2;

```

```

import Datos.DatosProductos;

```

```

import us.lsi.graphs.virtual.SimpleEdgeAction;

```

```

public record ProductosEdge(ProductosVertex source,
    ProductosVertex target, Integer action, Double weight)
    implements SimpleEdgeAction<ProductosVertex,
        Integer> {

```

```

    public static ProductosEdge of(ProductosVertex source,
        ProductosVertex target, Integer action) {
        Integer index = source.index();
        Double w = DatosProductos.getPrecioProducto(index) *
            action * 1.0;
        return new ProductosEdge(source, target, action, w);
    }

```

```

}

```

Vertex

```
package Ejercicio2;
```

```
import java.util.ArrayList;
```

```
import java.util.List;
```

```
import java.util.function.Predicate;
```

```
import java.util.stream.Collectors;
```

```
import Datos.DatosProductos;
```

```
import us.lsi.common.IntegerSet;
```

```
import us.lsi.common.List2;
```

```
import us.lsi.graphs.virtual.VirtualVertex;
```

```
public record ProductosVertex(Integer index, IntegerSet
categoriasPorCubrir, List<Integer> presupuestoRestante,
Integer acumValoracion) implements
VirtualVertex<ProductosVertex, ProductosEdge, Integer> {
```

```
    public static ProductosVertex initial() {
```

```
        IntegerSet categorias = IntegerSet
```

```
        .of(DatosProductos.getProductos().stream().map(v ->
v.categoria()).collect(Collectors.toSet()));
```

```
        List<Integer> presupuesto = List2.empty();
```

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

```
            presupuesto.add(DatosProductos.getPresupuesto());
```

```
        }
```

```
        return new ProductosVertex(0, categorias, presupuesto,
0);
```

```
    }
```

```
    public static Predicate<ProductosVertex> goal() {
        return v -> v.index() ==
DatosProductos.getNumProductos();
    }
```

```
    public static Predicate<ProductosVertex> goalHasSolution() {
        return v -> v.categoriasPorCubrir.isEmpty() &&
v.acumValoracion >= 0;
    }
```

```
@Override
```

```
    public List<Integer> actions() {
        if (index == DatosProductos.getNumProductos()) {
            return List2.empty();
        } else if ((categoriasPorCubrir.isEmpty() &&
acumValoracion >= 3)
||
presupuestoRestante.get(DatosProductos.getCategoria(index))
-
DatosProductos.getPrecioProducto(index) < 0) {
            return List2.of(0);
        } else if (index == (DatosProductos.getNumProductos() -
1)) {
            Integer valoracionFinal = acumValoracion +
(DatosProductos.getValoracionProducto(index) - 3);
```



```

        if (valoracionFinal < 0) {
            return List2.empty();
        } else if (categoriasPorCubrir.size() >= 2) {
            return List2.empty();
        } else if
        (presupuestoRestante.get(DatosProductos.getCategoria(index))
        -
        DatosProductos.getPrecioProducto(index) < 0) {
            return List2.of(0);
        } else if
        (categoriasPorCubrir.contains(DatosProductos.getCategoria(index)))
        {
            return List2.of(1);
        }
    }
    return List2.of(0, 1);
}

```

```

@Override
public ProductosVertex neighbor(Integer a) {
    Integer j = index + 1;
    IntegerSet categorias = categoriasPorCubrir.copy();
    List<Integer> presupuesto = new
    ArrayList<Integer>(presupuestoRestante());
    Integer acum = acumValoracion;
    if (a == 1) {

        categorias.remove(DatosProductos.getCategoria(index));

        presupuesto.set(DatosProductos.getCategoria(index),

```

```

        presupuesto.get(DatosProductos.getCategoria(index))
            -
        DatosProductos.getPrecioProducto(index));
        acum = acumValoracion +
        DatosProductos.getValoracionProducto(index) - 3;
    }
    return new ProductosVertex(j, categorias, presupuesto,
    acum);
}

```

@Override

```

public ProductosEdge edge(Integer a) {
    return ProductosEdge.of(this, neighbor(a), a);
}

```

```

public String toGraph() {
    return String.format("%s, %s, %s, %s", index(),
    this.categoriasPorCubrir.toString(),
    this.presupuestoRestante.toString(),
    this.acumValoracion.toString());
}

```

}

Heuristic

package Ejercicio2;

import java.util.function.Predicate;

import java.util.stream.IntStream;

```
import Datos.DatosProductos;
```

```
public class ProductosHeuristic {
```

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

```
        if (v1.categoriasPorCubrir().isEmpty()) {
```

```
            return 0.;
```

```
        }
```

```
        Integer index = v1.index();
```

```
        return IntStream.range(index,  
        DatosProductos.getNumProductos()).
```

```
            filter(i ->  
        v1.categoriasPorCubrir().contains(DatosProductos.getCategoria(i))).
```

```
        mapToDouble(DatosProductos::getPrecioProducto).sum();
```

```
    }
```

```
}
```

PD

```
package Ejercicio2_Manual;
```

```
import java.util.Comparator;
```

```
import java.util.List;
```

```
import java.util.Map;
```

```
import Datos.DatosProductos;
```

```
import Soluciones.SolucionProductos;
```

```
import us.lsi.common.List2;
```

```
import us.lsi.common.Map2;
```

```
public class ProductosPD {
```

```
    public static record SolucionParcial(Integer a, Integer weight)  
implements Comparable<SolucionParcial> {
```

```
        public static SolucionParcial of(Integer a, Integer weight) {  
            return new SolucionParcial(a, weight);  
        }  
    }
```

```
    @Override
```

```
    public int compareTo(SolucionParcial o) {  
        return this.weight.compareTo(o.weight);  
    }  
}
```

```
public static Map<ProductosProblem, SolucionParcial> memoria;  
public static Integer mejorValor = Integer.MAX_VALUE;
```

```
public static SolucionProductos search() {  
    memoria = Map2.empty();  
    mejorValor = Integer.MAX_VALUE;  
    pdr_search(ProductosProblem.initial(), 0);  
    return getSolucion();  
}
```

```
private static SolucionProductos getSolucion() {  
    List<Integer> acciones = List2.empty();  
    ProductosProblem prob = ProductosProblem.initial();
```

```

SolucionParcial spm = memoria.get(prob);
while (spm != null && spm.a != null) {
    ProductosProblem old = prob;
    acciones.add(spm.a);
    prob = old.neighbor(spm.a);
    spm = memoria.get(prob);
}
return SolucionProductos.of(acciones);
}

```

```

public static Integer acotar(Integer acumulado, ProductosProblem
origen, Integer action) {
    Integer weight =
DatosProductos.getPrecioProducto(origen.indice()) * action;
    return (int) (acumulado + weight +
origen.neighbor(action).heuristic());
}

```

```

private static SolucionParcial pdr_search(ProductosProblem
problema, Integer acumulado) {
    if(memoria.containsKey(problema)) {
        return memoria.get(problema);
    }
}

```

```

Boolean esGoal = ProductosProblem.goal().test(problema);
Boolean esSolucion =
ProductosProblem.goalHasSolution().test(problema);
if(esGoal && esSolucion) {
    SolucionParcial solucion_cb = new SolucionParcial(null, 0);
}

```

```

        memoria.put(problema, solucion_cb);
        if(acumulado < mejorValor) {
            mejorValor = acumulado;
        }
        return solucion_cb;
    }

```

```

List<SolucionParcial> solucionesPosibles = List2.empty();
for(Integer action : problema.actions()) {
    Integer estimacion = acotar(acumulado, problema, action);
    if(estimacion > mejorValor) {
        continue;
    }
    ProductosProblem vecino = problema.neighbor(action);
    Integer weight =
    DatosProductos.getPrecioProducto(problema.indice()) * action;
    SolucionParcial solucionVecino = pdr_search(vecino,
    acumulado + weight);
    if(solucionVecino != null) {
        SolucionParcial solucionBuena = new
        SolucionParcial(action, solucionVecino.weight() + weight);
        solucionesPosibles.add(solucionBuena);
    }
}

```

```

    SolucionParcial mejorSolucion =
    solucionesPosibles.stream().min(Comparator.naturalOrder()).orElse(
    null);
    if(mejorSolucion != null) {

```

```

        memoria.put(problema, mejorSolucion);
        return mejorSolucion;
    }
    return null;
}
}

```

Problem

```
package Ejercicio2_Manual;
```

```

import java.util.ArrayList;
import java.util.List;
import java.util.function.Predicate;
import java.util.stream.Collectors;
import java.util.stream.IntStream;

```

```

import Datos.DatosProductos;
import us.lsi.common.IntegerSet;
import us.lsi.common.List2;

```

```

public record ProductosProblem(Integer indice, IntegerSet
categoriasPorCubrir, List<Integer> presupuestoRestante,
    Integer acumValoracion) {
    public static ProductosProblem initial() {
        IntegerSet categorias =
            IntegerSet.of(DatosProductos.getProductos().stream()
                .map(obj -> obj.categoria())
                .collect(Collectors.toSet()));
        List<Integer> presupuesto = List2.empty();
    }
}

```

```

        for(int i = 0; i < categorias.size(); i++) {
            presupuesto.add(DatosProductos.getPresupuesto());
        }
        return new ProductosProblem(0, categorias,
presupuesto,
0);
    }

```

```

    public static Predicate<ProductosProblem> goal() {
        return v -> v.indice() ==
DatosProductos.getNumProductos();
    }

```

```

    public static Predicate<ProductosProblem> goalHasSolution() {
        return v -> v.categoriasPorCubrir().isEmpty() &&
v.acumValoracion >= 0;
    }

```

```

    public List<Integer> actions() {
        if (indice == DatosProductos.getNumProductos()) {
            return List2.empty();
        } else {
            if
(indice.equals(DatosProductos.getNumProductos() - 1)) {
                Integer valoracionFinal = acumValoracion +
DatosProductos.getValoracionProducto(indice) - 3;
                if (valoracionFinal < 0 ||
categoriasPorCubrir.size() > 1) {
                    return List2.of();
                }
            }
        }
    }

```



```

        } else if
        (presupuestoRestante.get(DatosProductos.getCategoria(indice))
        -
        DatosProductos.getPrecioProducto(indice) < 0) {
            return List2.of(0);
        } else if
        (categoriasPorCubrir.contains(DatosProductos.getCategoria(indice))
        ) {
            return List2.of(1);
        } else {
            return List2.of(0, 1);
        }
    } else if (categoriasPorCubrir.isEmpty())
    ||
    presupuestoRestante.get(DatosProductos.getCategoria(indice))
    -
    DatosProductos.getPrecioProducto(indice) < 0) {
        return List2.of(0);
    } else {
        return List2.of(0, 1);
    }
}
}

```

```

public Double heuristic() {
    if (this.categoriasPorCubrir().isEmpty())
        return 0.0;
    else
        return
        IntStream.range(this.indice(),
        DatosProductos.getNumProductos())

```

```

                                .filter(x                                ->
this.categoriasPorCubrir().contains(DatosProductos.getCategoria(x))
)

```

```

        .mapToDouble(DatosProductos::getPrecioProducto).min().orElse(
Double.MAX_VALUE);
    }

```

```

public ProductosProblem neighbor(Integer a) {
    Integer i = indice() + 1;
    Integer acum = acumValoracion();
    IntegerSet categorias = categoriasPorCubrir().copy();
    List<Integer> presupuestos = new
ArrayList<Integer>(presupuestoRestante());
    if (a == 1) {

        categorias.remove(DatosProductos.getCategoria(indice()));

        presupuestos.set(DatosProductos.getCategoria(indice()),

        presupuestos.get(DatosProductos.getCategoria(indice()))

        -
        DatosProductos.getPrecioProducto(indice()));
        acum = acumValoracion +
        DatosProductos.getValoracionProducto(indice) - 3;
    }
    return new ProductosProblem(i, categorias,
presupuestos, acum);
}
}

```

Test A*

```
package Tests;
```

```
import java.util.List;
```

```
import java.util.Locale;
```

```
import org.jgrapht.GraphPath;
```

```
import Datos.DatosProductos;
```

```
import Ejercicio2.ProductosEdge;
```

```
import Ejercicio2.ProductosHeuristic;
```

```
import Ejercicio2.ProductosVertex;
```

```
import Soluciones.SolucionProductos;
```

```
import us.lsi.colors.GraphColors;
```

```
import us.lsi.colors.GraphColors.Color;
```

```
import us.lsi.graphs.alg.AStar;
```

```
import us.lsi.graphs.virtual.EDGraph;
```

```
import us.lsi.graphs.virtual.EDGraph.Type;
```

```
import us.lsi.path.EDGraphPath.PathType;
```

```
public class Ejercicio2Test {
```

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

```
        Locale.setDefault(Locale.of("en", "US"));
```

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

```
DatosProductos.iniDatos("Ficheros/Ejercicio2DatosEntrada" + i  
+ ".txt");
```

```
System.out.println("\n\n>\tResultados para el test " + i + "\n");
```

```
ProductosVertex start = ProductosVertex.initial();
```

```
EGraph<ProductosVertex, ProductosEdge> grafo =  
    EGraph.virtual(start,      ProductosVertex.goal(),  
    PathType.Sum, Type.Min)  
    .edgeWeight(x -> x.weight())  
  
    .goalHasSolution(ProductosVertex.goalHasSolution())  
    .heuristic(ProductosHeuristic::heuristic)  
    .build();
```

```
System.out.println("\n\n#### Ejercicio 2 Algoritmo A* ####");
```

```
AStar<ProductosVertex, ProductosEdge, ?> astar =  
    AStar.of(grafo);
```

```
GraphPath<ProductosVertex, ProductosEdge> camino =  
    astar.search().get();
```

```
List<Integer> camino_as =  
    camino.getEdgeList().stream()  
    .map(x -> x.action())  
    .toList();
```

```
SolucionProductos sol = SolucionProductos.of(camino_as);  
System.out.println(sol);
```

```
GraphColors.toDot(astar.outGraph(),  
    "Grafos_Generados/Ejercicio2/Ejercicio2Auto" + i +  
".gv",  
    v -> v.toGraph(),  
    e -> e.action().toString() + ", " + e.weight().toString(),  
    v -> GraphColors.colorIf(Color.green,  
ProductosVertex.goal().test(v)),  
    e -> GraphColors.colorIf(Color.green,  
(camino.getEdgeList().contains(e))));  
}  
}  
  
}
```

```
Presupuesto: 150
Producto: 0, Precio: 10, Categoria: 0, Valoracion: 5
Producto: 1, Precio: 150, Categoria: 0, Valoracion: 4
Producto: 2, Precio: 150, Categoria: 1, Valoracion: 4
Producto: 3, Precio: 100, Categoria: 2, Valoracion: 2
Producto: 4, Precio: 120, Categoria: 2, Valoracion: 5
```

```
> Resultados para el test 1
```

```
#### Ejercicio 2 Algoritmo A* ####
{Producto0, Producto2, Producto3}
Precio Total:260.0
Presupuesto: 100
```

```
Producto: 0, Precio: 75, Categoria: 3, Valoracion: 5
Producto: 1, Precio: 10, Categoria: 2, Valoracion: 4
Producto: 2, Precio: 15, Categoria: 1, Valoracion: 2
Producto: 3, Precio: 50, Categoria: 0, Valoracion: 2
Producto: 4, Precio: 80, Categoria: 1, Valoracion: 5
Producto: 5, Precio: 95, Categoria: 2, Valoracion: 5
Producto: 6, Precio: 25, Categoria: 3, Valoracion: 4
```

```
> Resultados para el test 2
```

```
#### Ejercicio 2 Algoritmo A* ####
{Producto1, Producto2, Producto3, Producto6}
Precio Total:100.0
Presupuesto: 10
```

```
Producto: 0, Precio: 5, Categoria: 0, Valoracion: 2
Producto: 1, Precio: 4, Categoria: 2, Valoracion: 1
Producto: 2, Precio: 3, Categoria: 1, Valoracion: 3
Producto: 3, Precio: 5, Categoria: 3, Valoracion: 1
Producto: 4, Precio: 5, Categoria: 2, Valoracion: 1
Producto: 5, Precio: 8, Categoria: 4, Valoracion: 1
Producto: 6, Precio: 8, Categoria: 1, Valoracion: 2
Producto: 7, Precio: 7, Categoria: 0, Valoracion: 5
Producto: 8, Precio: 6, Categoria: 2, Valoracion: 4
Producto: 9, Precio: 10, Categoria: 1, Valoracion: 4
```

```
> Resultados para el test 3
```

```
#### Ejercicio 2 Algoritmo A* ####
{Producto3, Producto5, Producto7, Producto8, Producto9}
Precio Total:36.0
```

Test BT

package Tests;

```
import java.util.List;
import java.util.Locale;
import java.util.Optional;

import org.jgrapht.GraphPath;

import Datos.DatosProductos;
import Ejercicio2.ProductosEdge;
import Ejercicio2.ProductosHeuristic;
import Ejercicio2.ProductosVertex;
import Soluciones.SolucionProductos;
import us.lsi.colors.GraphColors;
import us.lsi.colors.GraphColors.Color;
import us.lsi.graphs.alg.BT;
import us.lsi.graphs.alg.GreedyOnGraph;
import us.lsi.graphs.virtual.EDGraph;
import us.lsi.graphs.virtual.EDGraph.Type;
import us.lsi.path.EDGraphPath.PathType;

public class Ejercicio2TestBT {

    public static void main(String[] args) {

        Locale.setDefault(Locale.of("en", "US"));

        for (int i = 1; i <= 3; i++) {
            DatosProductos.iniDatos("Ficheros/Ejercicio2DatosEntrada"
+ i + ".txt");
```

```
System.out.println("\n\n>\tResultados para el test " + i + "\n");
```

```
ProductosVertex start = ProductosVertex.initial();
```

```
EGraph<ProductosVertex, ProductosEdge> grafo =  
    EGraph.virtual(start, ProductosVertex.goal(),  
PathType.Sum, Type.Min)  
    .edgeWeight(x -> x.weight())  
    .goalHasSolution(ProductosVertex.goalHasSolution())  
    .heuristic(ProductosHeuristic::heuristic)  
    .build();
```

```
System.out.println("\n\n#### Ejercicio 1 Algoritmo BT ####");
```

```
Boolean conVoraz = false;
```

```
SolucionProductos sv = null;
```

```
Optional<GraphPath<ProductosVertex, ProductosEdge>> gp  
= Optional.empty();
```

```
if (conVoraz) {  
    GreedyOnGraph<ProductosVertex, ProductosEdge> ga =  
GreedyOnGraph.of(grafo);  
    gp = ga.search();  
    if (gp.isPresent()) sv = SolucionProductos.of(gp.get());  
    System.out.println("Sv = "+sv);  
}
```



```
BT<ProductosVertex, ProductosEdge, SolucionProductos>
bta = null;
```

```
    if(gp.isPresent())
        bta = BT.of(grafo, SolucionProductos.of,
gp.get().getWeight(), gp.get(), true);
    else
        bta = BT.of(grafo, null, null, null, true);
```

```
bta.search();
sv = SolucionProductos.of(bta.optimalPath().orElse(null));
```

```
List<ProductosEdge> le =
bta.optimalPath().get().getEdgeList();
```

```
System.out.println("Sol opt = "+sv);
```

```
var outGraph = bta.outGraph();
```

```
if(outGraph!=null)
    GraphColors.toDot(bta.outGraph(),
        "Grafos_Generados/Ejercicio2/Ejercicio2BT"+i+".gv",
        v -> v.toGraph(),
        e -> e.action().toString(),
        v -> GraphColors.colorIf(Color.red,
ProductosVertex.goal().test(v)),
        e -> GraphColors.colorIf(Color.red, le.contains(e)));
```

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

```

    }
}
}

```

```

Presupuesto: 150

Producto: 0, Precio: 10, Categoria: 0, Valoracion: 5
Producto: 1, Precio: 150, Categoria: 0, Valoracion: 4
Producto: 2, Precio: 150, Categoria: 1, Valoracion: 4
Producto: 3, Precio: 100, Categoria: 2, Valoracion: 2
Producto: 4, Precio: 120, Categoria: 2, Valoracion: 5

```

```

> Resultados para el test 1

#### Ejercicio 1 Algoritmo BT ####
Sol opt = {Producto0, Producto2, Producto3}
Precio Total:260.0

```

```

Presupuesto: 100

```

```

Producto: 0, Precio: 75, Categoria: 3, Valoracion: 5
Producto: 1, Precio: 10, Categoria: 2, Valoracion: 4
Producto: 2, Precio: 15, Categoria: 1, Valoracion: 2
Producto: 3, Precio: 50, Categoria: 0, Valoracion: 2
Producto: 4, Precio: 80, Categoria: 1, Valoracion: 5
Producto: 5, Precio: 95, Categoria: 2, Valoracion: 5
Producto: 6, Precio: 25, Categoria: 3, Valoracion: 4

```

```

> Resultados para el test 2

#### Ejercicio 1 Algoritmo BT ####
Sol opt = {Producto1, Producto2, Producto3, Producto6}
Precio Total:100.0

```

```

Presupuesto: 10

```

```

Producto: 0, Precio: 5, Categoria: 0, Valoracion: 2
Producto: 1, Precio: 4, Categoria: 2, Valoracion: 1
Producto: 2, Precio: 3, Categoria: 1, Valoracion: 3
Producto: 3, Precio: 5, Categoria: 3, Valoracion: 1
Producto: 4, Precio: 5, Categoria: 2, Valoracion: 1
Producto: 5, Precio: 8, Categoria: 4, Valoracion: 1
Producto: 6, Precio: 8, Categoria: 1, Valoracion: 2
Producto: 7, Precio: 7, Categoria: 0, Valoracion: 5
Producto: 8, Precio: 6, Categoria: 2, Valoracion: 4
Producto: 9, Precio: 10, Categoria: 1, Valoracion: 4

```

```

> Resultados para el test 3

#### Ejercicio 1 Algoritmo BT ####
Sol opt = {Producto3, Producto5, Producto7, Producto8, Producto9}
Precio Total:36.0

```

Test PDR

package Tests;

```
import java.util.Locale;
import java.util.Optional;
import java.util.function.Predicate;

import org.jgrapht.GraphPath;

import Datos.DatosProductos;
import Ejercicio2.ProductosEdge;
import Ejercicio2.ProductosHeuristic;
import Ejercicio2.ProductosVertex;
import Soluciones.SolucionProductos;
import us.lsi.colors.GraphColors;
import us.lsi.colors.GraphColors.Color;
import us.lsi.graphs.alg.GreedyOnGraph;
import us.lsi.graphs.alg.PDR;
import us.lsi.graphs.virtual.EDGraph;
import us.lsi.graphs.virtual.EDGraph.Type;
import us.lsi.path.EDGraphPath.PathType;

public class Ejercicio2TestPDR {
    public static void main(String[] args) {
        Locale.setDefault(Locale.of("en", "US"));

        for (Integer id_fichero = 1; id_fichero <= 3; id_fichero++) {

            DatosProductos.iniDatos("Ficheros/Ejercicio2DatosEntrada"+id_fichero+".txt");
```

```
ProductosVertex vInicial = ProductosVertex.initial();  
Predicate<ProductosVertex> goal = ProductosVertex.goal();
```

```
EGraph<ProductosVertex, ProductosEdge> graph =  
    EGraph.virtual(vInicial, goal, PathType.Sum, Type.Min)  
        .goalHasSolution(ProductosVertex.goalHasSolution())  
        .heuristic(ProductosHeuristic::heuristic)  
        .build();
```

```
GreedyOnGraph<ProductosVertex, ProductosEdge>  
alg_voraz = GreedyOnGraph.of(graph);  
GraphPath<ProductosVertex, ProductosEdge> path =  
alg_voraz.path();  
path = alg_voraz.isSolution(path)? path: null;
```

```
PDR<ProductosVertex, ProductosEdge, SolucionProductos>  
alg_pdr = path==null?  
    PDR.of(graph):  
    PDR.of(graph, null, path.getWeight(), path, true);
```

```
Optional<GraphPath<ProductosVertex, ProductosEdge>> gp  
= alg_pdr.search();  
var res = alg_pdr.search().orElse(null);  
var outGraph = alg_pdr.outGraph();
```

```
if(outGraph!=null) {  
    GraphColors.toDot(alg_pdr.outGraph,
```

```
"Grafos_Generados/Ejercicio2/Ejercicio2PDR"+id_fichero+".gv",
```

```

        v -> v.toGraph(),
        e -> e.action().toString(),
        v -> GraphColors.colorIf(Color.red,
ProductosVertex.goal().test(v)),
        e -> GraphColors.colorIf(Color.red,
gp.isPresent()?gp.get().getEdgeList().contains(e):false));
    }

    if(res!=null)
        System.out.println("Solucion PDR: " +
SolucionProductos.of(res) + "\n");
    else
        System.out.println("PDR no obtuvo solucion\n");
    }
}
}

```

```

C:\Program Files\Java\jdk-20\bin\javaw.exe (12 May 20
Presupuesto: 150

Producto: 0, Precio: 10, Categoria: 0, Valoracion: 5
Producto: 1, Precio: 150, Categoria: 0, Valoracion: 4
Producto: 2, Precio: 150, Categoria: 1, Valoracion: 4
Producto: 3, Precio: 100, Categoria: 2, Valoracion: 2
Producto: 4, Precio: 120, Categoria: 2, Valoracion: 5

Solucion PDR: {Producto0, Producto2, Producto3}
Precio Total:260.0

```

```

Presupuesto: 100

Producto: 0, Precio: 75, Categoria: 3, Valoracion: 5
Producto: 1, Precio: 10, Categoria: 2, Valoracion: 4
Producto: 2, Precio: 15, Categoria: 1, Valoracion: 2
Producto: 3, Precio: 50, Categoria: 0, Valoracion: 2
Producto: 4, Precio: 80, Categoria: 1, Valoracion: 5
Producto: 5, Precio: 95, Categoria: 2, Valoracion: 5
Producto: 6, Precio: 25, Categoria: 3, Valoracion: 4

Solucion PDR: {Producto1, Producto2, Producto3, Producto6}
Precio Total:100.0

```

```

Producto: 0, Precio: 5, Categoria: 0, Valoracion: 2
Producto: 1, Precio: 4, Categoria: 2, Valoracion: 1
Producto: 2, Precio: 3, Categoria: 1, Valoracion: 3
Producto: 3, Precio: 5, Categoria: 3, Valoracion: 1
Producto: 4, Precio: 5, Categoria: 2, Valoracion: 1
Producto: 5, Precio: 8, Categoria: 4, Valoracion: 1
Producto: 6, Precio: 8, Categoria: 1, Valoracion: 2
Producto: 7, Precio: 7, Categoria: 0, Valoracion: 5
Producto: 8, Precio: 6, Categoria: 2, Valoracion: 4
Producto: 9, Precio: 10, Categoria: 1, Valoracion: 4

Solucion PDR: {Producto3, Producto5, Producto7, Producto8, Producto9}
Precio Total:36.0

```

Test Manual

```
package Tests_Manuales;
```

```
import java.util.List;
```

```
import Datos.DatosProductos;
```

```
import Ejercicio2_Manual.ProductosPD;
```

```
import us.lsi.common.String2;
```

```
public class Ejercicio2Test_Manual {
```

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

```
        List.of(1,2,3).forEach(num_test -> {
```

```
            System.out.println("*****
FICHERO " + num_test + "*****");
```

```
            DatosProductos.iniDatos("Ficheros/Ejercicio2DatosEntrada"+n
um_test+".txt");
```

```
                String2.toConsole("Solucion del ejercicio 2 usando
programacion dinamica manual para el fichero %s: %s\n", num_test,
ProductosPD.search());
```

```
        });
```

```
    }
```

}

```
terminated> Ejercicio2TestManual para Application\src\Program Files\java\jdk-20\bin\javaw.exe (12 May 2024 10:20:37) [pid: 16304]
*****FICHERO 1*****
Presupuesto: 150

Producto: 0, Precio: 10, Categoria: 0, Valoracion: 5
Producto: 1, Precio: 150, Categoria: 0, Valoracion: 4
Producto: 2, Precio: 150, Categoria: 1, Valoracion: 4
Producto: 3, Precio: 100, Categoria: 2, Valoracion: 2
Producto: 4, Precio: 120, Categoria: 2, Valoracion: 5

Solucion del ejercicio 2 usando programacion dinamica manual para el fichero 1: {Producto0, Producto2, Producto3}
Precio Total:260.0

*****FICHERO 2*****
Presupuesto: 100
```

```
Producto: 0, Precio: 75, Categoria: 3, Valoracion: 5
Producto: 1, Precio: 10, Categoria: 2, Valoracion: 4
Producto: 2, Precio: 15, Categoria: 1, Valoracion: 2
Producto: 3, Precio: 50, Categoria: 0, Valoracion: 2
Producto: 4, Precio: 80, Categoria: 1, Valoracion: 5
Producto: 5, Precio: 95, Categoria: 2, Valoracion: 5
Producto: 6, Precio: 25, Categoria: 3, Valoracion: 4

Solucion del ejercicio 2 usando programacion dinamica manual para el fichero 2: {Producto1, Producto2, Producto3}
Precio Total:100.0
```

```
*****FICHERO 3*****
Presupuesto: 10

Producto: 0, Precio: 5, Categoria: 0, Valoracion: 2
Producto: 1, Precio: 4, Categoria: 2, Valoracion: 1
Producto: 2, Precio: 3, Categoria: 1, Valoracion: 3
Producto: 3, Precio: 5, Categoria: 3, Valoracion: 1
Producto: 4, Precio: 5, Categoria: 2, Valoracion: 1
Producto: 5, Precio: 8, Categoria: 4, Valoracion: 1
Producto: 6, Precio: 8, Categoria: 1, Valoracion: 2
Producto: 7, Precio: 7, Categoria: 0, Valoracion: 5
Producto: 8, Precio: 6, Categoria: 2, Valoracion: 4
Producto: 9, Precio: 10, Categoria: 1, Valoracion: 4

Solucion del ejercicio 2 usando programacion dinamica manual para el fichero 3: {Producto3, Producto5, Producto9}
Precio Total:36.0
```

Ejercicio 3

Datos

```
package Datos;
```

```
import java.util.List;
```

```
import us.lsi.common.Files2;
```

```
import us.lsi.common.List2;
```

```
public class DatosDistribuidor {
```

```
    public static List<Destino> destinos;
```

```
    public static List<Producto> productos;
```

```
    public record Destino(Integer codigo, Integer demandaMinima)  
{
```

```
        public static Destino create(String s) {
```

```
            String[] s1 = s.split(";");
```

```
            String[] d = s1[0].trim().split(": demandaminima=");
```

```
                Integer                codigo                =  
Integer.valueOf(d[0].trim().replace("D", ""));
```

```
                Integer                demandaMinima        =  
Integer.valueOf(d[1].trim());
```

```
            return new Destino(codigo, demandaMinima);
```

```
        }
```

```
    }
```

```
    public record Producto(Integer codigo, Integer unidades,  
List<Coste> costes) {
```



```

        public static Producto create(String s) {
            String[] s1 = s.split(";");
            String[] s2 = s1[0].trim().split(" -> uds=");
            Integer          codigo          =
Integer.valueOf(s2[0].trim().replace("P", ""));
            Integer unidades = Integer.valueOf(s2[1].trim());
            String[]          costeStr          =
s1[1].trim().split("coste_almacenamiento=");
            List<Coste> costes = List2.parse(costeStr[1], ",",
Coste::create);

            return new Producto(codigo, unidades, costes);
        }
    }

```

```

    public record Coste(Integer destino, Integer coste) {
        public static Coste create(String s) {
            String[] s1 = s.replace("(", "").replace(")",
"").split(":");
            Integer destino = Integer.valueOf(s1[0].trim());
            Integer coste = Integer.valueOf(s1[1].trim());
            return new Coste(destino, coste);
        }
    }

```

```

    public static void iniDatos(String fichero) {
        destinos = Files2.streamFromFile(fichero).filter(x ->
x.contains("demandaminima")).map(Destino::create)
            .toList();
    }

```

```
        productos = Files2.streamFromFile(fichero).filter(x ->
x.contains(" -> uds=")).map(Producto::create).toList();
    }
```

```
public static void main(String[] args) {
    iniDatos("ficheros/Ejercicio3DatosEntrada3.txt");
}
```

```
public static List<Destino> getDestinos() {
    return destinos;
}
```

```
public static List<Coste> getCostes(Integer i) {
    return productos.get(i).costes();
}
```

```
public static List<Producto> getProductos() {
    return productos;
}
```

```
public static Integer getNumProductos() {
    return productos.size();
}
```

```
public static Integer getNumDestinos() {
    return destinos.size();
}
```

```
public static Integer getDemanda(Integer i) {  
    return destinos.get(i).demandaMinima();  
}
```

```
public static List<Integer> getUnidades() {  
    return  
productos.stream().map(Producto::unidades).toList();  
}
```

```
public static Integer getUnidades(Integer i) {  
    return productos.get(i).unidades();  
}
```

```
public static List<Integer> getDemandas() {  
    return  
destinos.stream().map(Destino::demandaMinima).toList();  
}
```

```
public static Integer getCoste(Integer i, Integer j) {  
    return productos.get(i).costes().get(j).coste();  
}
```

```
public static Integer getUnidad(Integer i) {  
    return productos.get(i).unidades();  
  
}  
}
```

Solucion

```
package Soluciones;
```

```
import java.util.ArrayList;
```

```
import java.util.HashMap;
```

```
import java.util.List;
```

```
import java.util.Map;
```

```
import java.util.stream.Collectors;
```

```
import org.jgrapht.GraphPath;
```

```
import Datos.DatosDistribuidor;
```

```
import Ejercicio3.DistribuidorEdge;
```

```
import Ejercicio3.DistribuidorVertex;
```

```
public record SolucionDistribuidor(Double peso, Map<Integer,  
List<Integer>> reparto)
```

```
    implements Comparable<SolucionDistribuidor> {
```

```
        public static SolucionDistribuidor  
of(GraphPath<DistribuidorVertex, DistribuidorEdge> path) {
```

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

```
            return SolucionDistribuidor.of(la);
```

```
        }
```

```
        public static SolucionDistribuidor of(List<Integer> value) {
```

```
            Double costeAcum = 0.;
```

```
            Map<Integer, List<Integer>> s = new HashMap<>();
```

```

        for (int x = 0; x < value.size(); x++) {
            Integer indiceDestino = x %
DatosDistribuidor.getNumDestinos();
            Integer indiceProducto = x /
DatosDistribuidor.getNumDestinos();
            if (s.containsKey(indiceProducto)) {
                s.get(indiceProducto).add(value.get(x));
            } else {
                List<Integer> ls = new ArrayList<>();
                ls.add(value.get(x));
                s.put(indiceProducto, ls);
            }
            costeAcum += value.get(x) *
DatosDistribuidor.getCoste(indiceProducto, indiceDestino);
        }
        return new SolucionDistribuidor(costeAcum, s);
    }

```

@Override

```

public int compareTo(SolucionDistribuidor o) {
    return this.peso().compareTo(o.peso());
}

```

@Override

```

public String toString() {
    String s = this.reparto().entrySet().stream()
        .map(e -> "Cantidad de producto" + e.getKey()
+ " en el destino" + e.getValue() + ": ")

```

```

        +
e.getValue().stream().mapToInt(Integer::intValue).sum())
        .collect(Collectors.joining("\n", "Productos
enviados a destinos: \n",
        "\n Coste total: " + this.peso() + "
\n"));
    return s;
}

```

```

}

```

Edge

```

package Ejercicio3;

```

```

import Datos.DatosDistribuidor;

```

```

import us.lsi.graphs.virtual.SimpleEdgeAction;

```

```

public record DistribuidorEdge(DistribuidorVertex source,
DistribuidorVertex target, Integer action, Double weight)

```

```

    implements SimpleEdgeAction<DistribuidorVertex,
Integer> {

```

```

        public static DistribuidorEdge of(DistribuidorVertex origen,
DistribuidorVertex destino, Integer accion) {

```

```

            Integer pro = origen.z() /
DatosDistribuidor.getNumDestinos();

```

```

            Integer des = origen.z() %
DatosDistribuidor.getNumDestinos();

```

```

            Double w = DatosDistribuidor.getCoste(pro, des) * accion
* 1.0;

```

```

            return new DistribuidorEdge(origen, destino, accion, w);

```

```

        }

```

```

}

```

Vertex

```
package Ejercicio3;
```

```
import java.util.ArrayList;
```

```
import java.util.List;
```

```
import java.util.function.Predicate;
```

```
import Datos.DatosDistribuidor;
```

```
import us.lsi.common.List2;
```

```
import us.lsi.graphs.virtual.VirtualVertex;
```

```
public record DistribuidorVertex(Integer z, List<Integer> unidadesRestantes, List<Integer> demandasRestantes)
```

```
    implements VirtualVertex<DistribuidorVertex, DistribuidorEdge, Integer> {
```

```
    public static DistribuidorVertex of(Integer z, List<Integer> unidadesRestantes, List<Integer> demandasRestantes) {
```

```
        return new DistribuidorVertex(z, unidadesRestantes, demandasRestantes);
```

```
    }
```

```
    public static DistribuidorVertex inicial() {
```

```
        List<Integer> unidadesIniciales = new ArrayList<>();
```

```
        List<Integer> demandasIniciales = new ArrayList<>();
```

```
        for(int i = 0; i<DatosDistribuidor.getNumProductos(); i++) {
```

```
            unidadesIniciales.add(DatosDistribuidor.getUnidad(i));
```

```
        }
```

```
        for(int j = 0; j<DatosDistribuidor.getNumDestinos(); j++) {
```

```

        demandasIniciales.add(DatosDistribuidor.getDemanda(j));
    }
    return of(0, unidadesIniciales, demandasIniciales);
}

```

```

public static Predicate<DistribuidorVertex> goal(){
    return x -> x.z() ==
DatosDistribuidor.getNumDestinos()*DatosDistribuidor.getNumProd
uctos();
}

```

```

public static Predicate<DistribuidorVertex> goalHasSolution() {
    return x-> x.demandasRestantes().stream().allMatch(y -> y ==
0);
}

```

```

@Override
public List<Integer> actions() {
    List<Integer> alternativas = new ArrayList<>();
    Integer numVariables = DatosDistribuidor.getNumProductos() *
DatosDistribuidor.getNumDestinos();
    if(z() >= numVariables) {
        return alternativas;
    }
    else {
        Integer indiceDestino =
z()%DatosDistribuidor.getNumDestinos();
        Integer indiceProducto =
z()/DatosDistribuidor.getNumDestinos();
    }
}

```



```

Integer uniRes = unidadesRestantes().get(indiceProducto);
Integer demRes = demandasRestantes().get(indiceDestino);
if(demRes == 0 || uniRes == 0) {
    alternativas = List2.of(0);
}
else if(uniRes < 0) {
    return List2.empty();
}
else if(uniRes < demRes) {
    alternativas = List2.of(0, uniRes);
} else {
    alternativas = List2.of(0, demRes);
}
return alternativas;
}
}

```

@Override

```

public DistribuidorVertex neighbor(Integer a) {
    Integer i = z() / DatosDistribuidor.getNumDestinos();
    Integer j = z() % DatosDistribuidor.getNumDestinos();

```

```

    List<Integer> unidadesRAct = new
ArrayList<>(unidadesRestantes());

```

```

    List<Integer> demandasRAct = new
ArrayList<>(demandasRestantes());

```

```

    Integer indiceProducto = i;

```

```

        Integer indiceDestino = j;

        unidadesRAct.set(indiceProducto,
        unidadesRAct.get(indiceProducto) - a);

        demandasRAct.set(indiceDestino,
        demandasRAct.get(indiceDestino) - a);

        return of(z() + 1, unidadesRAct, demandasRAct);
    }

    @Override
    public DistribuidorEdge edge(Integer a) {
        return DistribuidorEdge.of(this, neighbor(a), a);
    }

    public String toGraph() {
        return String.format("%d;      %s;      %s",      this.z,
        this.unidadesRestantes, this.demandasRestantes);
    }
}

```

Heuristic

```
package Ejercicio3;
```

```
import java.util.function.Predicate;
```

```
import java.util.stream.IntStream;
```

```
import Datos.DatosDistribuidor;
```

```

public class DistribuidorHeuristic {
    public static Double heuristic(DistribuidorVertex v1,
    Predicate<DistribuidorVertex> goal, DistribuidorVertex v2) {
        if (v1.demandasRestantes().stream().allMatch(x -> x <=
0))
            return 0.;

        Integer ultimoIndice =
DatosDistribuidor.getNumDestinos() *
DatosDistribuidor.getNumProductos();

        return IntStream.range(v1.z(), ultimoIndice)
            .filter(i -> v1.demandasRestantes().get(i %
DatosDistribuidor.getNumDestinos()) > 0)
            .mapToDouble(i ->
DatosDistribuidor.getCoste(i / DatosDistribuidor.getNumDestinos(),
i %
DatosDistribuidor.getNumDestinos()))
            .min().orElse(Double.MAX_VALUE);
    }
}

```

BT

```
package Ejercicio3_Manual;
```

```
import Soluciones.SolucionDistribuidor;
```

```

public class DistribuidorBT {
    private static Double mejorValor;
    private static DistribuidorState estado;
    private static SolucionDistribuidor solucion;

```

```

public static void search() {
    solucion = null;
    mejorValor = Double.MAX_VALUE; // Estamos minimizando
    estado = DistribuidorState.initial();
    bt_search();
}

private static void bt_search() {
    if (estado.esSolucion()) {
        Double valorObtenido = estado.acumulado;
        if (valorObtenido < mejorValor) { // Estamos minimizando
            mejorValor = valorObtenido;
            solucion = estado.getSolucion();
        }
    } else if (!estado.esTerminal()){
        for (Integer a: estado.actual.actions()) {
            Double cota = estado.cota(a);
            if (cota >= mejorValor) continue; // Estamos minimizando
            estado.forward(a);
            bt_search();
            estado.back();
        }
    }
}

public static SolucionDistribuidor getSolucion() {
    return solucion;
}

```

```
}  
}
```

Problem

```
package Ejercicio3_Manual;
```

```
import java.util.ArrayList;  
import java.util.List;  
import java.util.function.Predicate;  
import java.util.stream.IntStream;
```

```
import Datos.DatosDistribuidor;  
import us.lsi.common.List2;
```

```
public record DistribuidorProblem(Integer z, List<Integer>  
unidadesRestantes, List<Integer> demandasRestantes) {
```

```
    public static DistribuidorProblem initial() {  
        List<Integer> uniRes = DatosDistribuidor.getProductos()  
            .stream()  
            .map(pro -> pro.unidades())  
            .toList();  
        List<Integer> demRes = DatosDistribuidor.getDestinos()  
            .stream()  
            .map(des -> des.demandaMinima())  
            .toList();  
        return new DistribuidorProblem(0, uniRes, demRes);  
    }
```

```
    public static Predicate<DistribuidorProblem> goal(){
```

```

        return obj -> obj.z == DatosDistribuidor.getNumProductos() *
DatosDistribuidor.getNumDestinos();
    }

```

```

    public static Predicate<DistribuidorProblem> goalHasSolution(){
        return obj -> obj.demandasRestantes().stream().allMatch(x -> x
<= 0);
    }

```

```

    public List<Integer> actions() {
        List<Integer> alternativas = new ArrayList<>();

        Integer numVariables = DatosDistribuidor.getNumProductos() *
DatosDistribuidor.getNumDestinos();

        if(this.z >= numVariables) {
            return alternativas;
        }
        else {
            Integer indiceDestino = this.z %
DatosDistribuidor.getNumDestinos();

            Integer indiceProducto = this.z /
DatosDistribuidor.getNumDestinos();

            Integer uniRes =
this.unidadesRestantes.get(indiceProducto);

            Integer demRes =
this.demandasRestantes.get(indiceDestino);

            if(demRes == 0 || uniRes == 0) {
                alternativas = List2.of(0);
            }

            else if(uniRes < 0) {
                return List2.empty();
            }
        }
    }

```

```

    }
    else if(uniRes < demRes) {
        alternativas = List2.rangeList(0, uniRes + 1);
    } else {
        alternativas = List2.rangeList(0, demRes + 1);
    }
    return alternativas;
}
}

```

```

public DistribuidorProblem neighbor(Integer a) {
    Integer index = this.z + 1;
    List<Integer> unidades = List2.copy(this.unidadesRestantes);
    List<Integer> demandas = List2.copy(this.demandasRestantes);
    unidades.set(this.z / DatosDistribuidor.getNumDestinos(),
unidades.get(this.z / DatosDistribuidor.getNumDestinos()) - a);
    demandas.set(this.z % DatosDistribuidor.getNumDestinos(),
demandas.get(this.z % DatosDistribuidor.getNumDestinos()) - a);
    return new DistribuidorProblem(index, unidades, demandas);
}

```

```

public Double heuristic() {
    if(this.demandasRestantes().stream().allMatch(x -> x <= 0))
return 0.;
    else {
        Integer ultimoIndice = DatosDistribuidor.getNumDestinos() *
DatosDistribuidor.getNumProductos();
        return IntStream.range(this.z(), ultimoIndice)

```

```

        .mapToDouble(i -> DatosDistribuidor.getCoste(i) /
DatosDistribuidor.getNumDestinos(), i %)
        DatosDistribuidor.getNumDestinos()))
        .min()
        .orElse(Double.MAX_VALUE);
    }
}

```

```

    public static void main(String[] args) {

        DatosDistribuidor.iniDatos("ficheros/Ejercicio3DatosEntrada1.txt");
        System.out.println(initial());
    }
}

```

State

```
package Ejercicio3_Manual;
```

```
import java.util.List;
```

```
import Datos.DatosDistribuidor;
```

```
import Soluciones.SolucionDistribuidor;
```

```
import us.lsi.common.List2;
```

```

public class DistribuidorState {
    DistribuidorProblem actual;
    Double acumulado;
    List<Integer> acciones;
    List<DistribuidorProblem> anteriores;
}

```



```

    private DistribuidorState(DistribuidorProblem problema, Double
acum, List<Integer> lisA, List<DistribuidorProblem> lisP) {
        actual = problema;
        acumulado = acum;
        acciones = lisA;
        anteriores = lisP;
    }

```

```

    public static DistribuidorState of(DistribuidorProblem problema,
Double acum, List<Integer> lisA, List<DistribuidorProblem> lisP) {
        return new DistribuidorState(problema, acum, lisA, lisP);
    }

```

```

    public static DistribuidorState initial() {
        DistribuidorProblem inicio = DistribuidorProblem.initial();
        return of(inicio, 0., List2.empty(), List2.empty());
    }

```

```

    public void forward(Integer a) {
        acumulado += a * DatosDistribuidor.getCoste(actual.z() /
DatosDistribuidor.getNumDestinos(),          actual.z()          %)
DatosDistribuidor.getNumDestinos());
        acciones.add(a);
        anteriores.add(actual);
        actual = actual.neighbor(a);
    }

```

```

    public void back() {

```

```

        int ultimo = acciones.size() - 1;
        var problemaAnterior = anteriores.get(ultimo);
        acumulado -= acciones.get(ultimo) *
        DatosDistribuidor.getCoste(problemaAnterior.z()) /
        DatosDistribuidor.getNumDestinos(), problemaAnterior.z() %
        DatosDistribuidor.getNumDestinos());
        acciones.remove(ultimo);
        anteriores.remove(ultimo);
        actual = problemaAnterior;
    }

```

```

    public List<Integer> alternativas(){
        return actual.actions();
    }

```

```

    public Double cota(Integer a) {
        Double weight = a * DatosDistribuidor.getCoste(actual.z()) /
        DatosDistribuidor.getNumDestinos(), actual.z() %
        DatosDistribuidor.getNumDestinos()) * 1.;
        return acumulado + weight + actual.neighbor(a).heuristic();
    }

```

```

    public Boolean esTerminal() {
        return DistribuidorProblem.goal().test(actual);
    }

```

```

    public Boolean esSolucion() {
        return DistribuidorProblem.goalHasSolution().test(actual);
    }

```

```
    public SolucionDistribuidor getSolucion() {  
        return SolucionDistribuidor.of(acciones);  
    }  
}
```

Test A*

```
    package Tests;  
  
import java.util.List;  
import java.util.Locale;  
import java.util.function.Predicate;  
  
import org.jgrapht.GraphPath;  
  
import Datos.DatosDistribuidor;  
import Ejercicio3.DistribuidorEdge;  
import Ejercicio3.DistribuidorHeuristic;  
import Ejercicio3.DistribuidorVertex;  
import Soluciones.SolucionDistribuidor;  
import us.lsi.colors.GraphColors;  
import us.lsi.colors.GraphColors.Color;  
import us.lsi.graphs.alg.AStar;  
import us.lsi.graphs.virtual.EDGraph;  
import us.lsi.graphs.virtual.EDGraph.Type;  
import us.lsi.path.EDGraphPath.PathType;  
  
public class Ejercicio3Test {
```

```

public static void main(String[] args) {

    Locale.setDefault(Locale.of("en", "US"));

    for (int i = 1; i <= 3; i++) {
        DatosDistribuidor.iniDatos("Ficheros/Ejercicio3DatosEntrada"
+ i + ".txt");
        System.out.println("\n\n>\tResultados para el test " + i + "\n");

        DistribuidorVertex start = DistribuidorVertex.inicial();
        Predicate<DistribuidorVertex> goal = DistribuidorVertex.goal();

        EGraph<DistribuidorVertex, DistribuidorEdge> grafo =
            EGraph.virtual(start, goal, PathType.Sum,
Type.Min)
            .edgeWeight(x -> x.weight())

            .goalHasSolution(DistribuidorVertex.goalHasSolution())
            .heuristic(DistribuidorHeuristic::heuristic)
            .build();

        System.out.println("\n\n#### Ejercicio 3 Algoritmo A* ####");

        AStar<DistribuidorVertex, DistribuidorEdge, ?> astar =
            AStar.of(grafo);

        GraphPath<DistribuidorVertex, DistribuidorEdge> camino =

```

```
    astar.search().get();
```

```
    List<Integer> camino_as =  
        camino.getEdgeList().stream()  
            .map(x -> x.action())  
            .toList();
```

```
    SolucionDistribuidor sol = SolucionDistribuidor.of(camino_as);  
    System.out.println(sol);
```

```
    GraphColors.toDot(astar.outGraph(),  
        "Grafos_Generados/Ejercicio3/Ejercicio3Auto" + i +  
        ".gv",  
        v -> v.toGraph(),  
        e -> e.action().toString() + ", " + e.weight().toString(),  
        v -> GraphColors.colorIf(Color.green,  
        DistribuidorVertex.goal().test(v)),  
        e -> GraphColors.colorIf(Color.green,  
        (camino.getEdgeList().contains(e))));  
    }  
}
```

```

terminado: Ejercicio3Test.java Application C:\Program Files\Java\jdk-20\bin\java.exe (12 ms) 2024-10-26 19:16:20.207
> Resultados para el test 1

#### Ejercicio 3 Algoritmo A* ####
Productos enviados a destinos:
Cantidad de producto0 en el destino[5, 0, 0, 0, 0]: 5
Cantidad de producto1 en el destino[1, 3, 8, 10, 5]: 27
Coste total: 67.0

> Resultados para el test 2

#### Ejercicio 3 Algoritmo A* ####
Productos enviados a destinos:
Cantidad de producto0 en el destino[1, 3, 0, 0, 0, 0, 2]: 6
Cantidad de producto1 en el destino[0, 0, 2, 6, 4, 0, 0]: 12
Cantidad de producto2 en el destino[0, 0, 0, 0, 0, 8, 0]: 8
Coste total: 26.0

> Resultados para el test 3

#### Ejercicio 3 Algoritmo A* ####
Productos enviados a destinos:
Cantidad de producto0 en el destino[3, 0, 0, 5, 0, 0, 0, 0, 2, 2]: 12
Cantidad de producto1 en el destino[0, 0, 0, 0, 1, 3, 1, 10, 0, 0]: 15
Cantidad de producto2 en el destino[0, 1, 8, 0, 0, 0, 0, 0, 0, 4]: 13
Coste total: 40.0

```

Test BT

package Tests;

import java.util.List;

import java.util.Locale;

import java.util.Optional;

import org.jgrapht.GraphPath;

import Datos.DatosDistribuidor;

import Ejercicio3.DistribuidorEdge;

import Ejercicio3.DistribuidorHeuristic;

```

import Ejercicio3.DistribuidorVertex;
import Soluciones.SolucionDistribuidor;
import us.lsi.colors.GraphColors;
import us.lsi.colors.GraphColors.Color;
import us.lsi.graphs.alg.BT;
import us.lsi.graphs.alg.GreedyOnGraph;
import us.lsi.graphs.virtual.EDGraph;
import us.lsi.graphs.virtual.EDGraph.Type;
import us.lsi.path.EDGraphPath.PathType;

public class Ejercicio3TestBT {

    public static void main(String[] args) {

        Locale.setDefault(Locale.of("en", "US"));

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

            DatosDistribuidor.iniDatos("Ficheros/Ejercicio3DatosEntrada" + i +
            ".txt");

            System.out.println("\n\n>\tResultados para el test " + i + "\n");

            DistribuidorVertex start = DistribuidorVertex.inicial();

            EDGraph<DistribuidorVertex, DistribuidorEdge> grafo =
                EDGraph.virtual(start, DistribuidorVertex.goal(),
                PathType.Sum, Type.Min)
                .edgeWeight(x -> x.weight())

```

```
.goalHasSolution(DistribuidorVertex.goalHasSolution())  
.heuristic(DistribuidorHeuristic::heuristic)  
.build();
```

```
System.out.println("\n\n#### Ejercicio 3 Algoritmo BT ####");
```

```
Boolean conVoraz = false;
```

```
SolucionDistribuidor sv = null;
```

```
Optional<GraphPath<DistribuidorVertex, DistribuidorEdge>>  
gp = Optional.empty();
```

```
if (conVoraz) {  
    GreedyOnGraph<DistribuidorVertex, DistribuidorEdge> ga  
= GreedyOnGraph.of(grafo);  
    gp = ga.search();  
    if (gp.isPresent()) sv = SolucionDistribuidor.of(gp.get());  
    System.out.println("Sv = "+sv);  
}
```

```
BT<DistribuidorVertex, DistribuidorEdge,  
SolucionDistribuidor> bta = null;
```

```
if(gp.isPresent())  
    bta = BT.of(grafo, SolucionDistribuidor::of,  
gp.get().getWeight(), gp.get(), true);  
else  
    bta = BT.of(grafo, null, null, null, true);
```



```

    bta.search();

    sv = SolucionDistribuidor.of(bta.optimalPath().orElse(null));

    List<DistribuidorEdge> le =
    bta.optimalPath().get().getEdgeList();

    System.out.println("Sol opt = "+sv);

    var outGraph = bta.outGraph();

    if(outGraph!=null)
        GraphColors.toDot(bta.outGraph(),
            "Grafos_Generados/Ejercicio3/Ejercicio3BT"+i+".gv",
            v -> v.toGraph(),
            e -> e.action().toString(),
            v -> GraphColors.colorIf(Color.red,
DistribuidorVertex.goal().test(v)),
            e -> GraphColors.colorIf(Color.red, le.contains(e)));

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

```

```
terminated> Ejercicio3TestBT [Java Application] C:\Program Files\Java\jdk-20\bin\javaw.exe (12 may 2
```

```
> Resultados para el test 1
```

```
#### Ejercicio 3 Algoritmo BT ####
```

```
Sol opt = Productos enviados a destinos:
```

```
Cantidad de producto0 en el destino[5, 0, 0, 0, 0]: 5
```

```
Cantidad de producto1 en el destino[1, 3, 8, 10, 5]: 27
```

```
Coste total: 67.0
```

```
> Resultados para el test 2
```

```
#### Ejercicio 3 Algoritmo BT ####
```

```
Sol opt = Productos enviados a destinos:
```

```
Cantidad de producto0 en el destino[1, 3, 0, 0, 0, 0, 2]: 6
```

```
Cantidad de producto1 en el destino[0, 0, 2, 6, 4, 0, 0]: 12
```

```
Cantidad de producto2 en el destino[0, 0, 0, 0, 0, 8, 0]: 8
```

```
Coste total: 26.0
```

```
> Resultados para el test 3
```

```
#### Ejercicio 3 Algoritmo BT ####
```

```
Sol opt = Productos enviados a destinos:
```

```
Cantidad de producto0 en el destino[0, 0, 0, 5, 0, 0, 0, 2, 5]: 12
```

```
Cantidad de producto1 en el destino[0, 0, 0, 0, 1, 3, 1, 10, 0, 0]: 15
```

```
Cantidad de producto2 en el destino[3, 1, 8, 0, 0, 0, 0, 0, 0, 1]: 13
```

```
Coste total: 40.0
```

Test PDR

package Tests;

import java.util.List;

import java.util.Locale;

import java.util.stream.Collectors;

import org.jgrapht.GraphPath;

```

import Datos.DatosDistribuidor;
import Ejercicio3.DistribuidorEdge;
import Ejercicio3.DistribuidorHeuristic;
import Ejercicio3.DistribuidorVertex;
import Soluciones.SolucionDistribuidor;
import us.lsi.graphs.alg.PDR;
import us.lsi.graphs.virtual.EDGraph;
import us.lsi.graphs.virtual.EDGraph.Type;
import us.lsi.path.EDGraphPath.PathType;

public class Ejercicio3TestPDR {
    public static void main(String[] args) {
        Locale.setDefault(Locale.of("en", "US"));

        for (Integer id_fichero = 1; id_fichero <= 3; id_fichero++) {

            DatosDistribuidor.iniDatos("Ficheros/Ejercicio3DatosEntrada"+id_fic
            hero+".txt");

            System.out.println("\n\n>\tResultados para el test " +
            id_fichero + "\n");

            DistribuidorVertex vInicial = DistribuidorVertex.inicial();

            EDGraph<DistribuidorVertex, DistribuidorEdge> graph =
                EDGraph.virtual(vInicial, DistribuidorVertex.goal(),
                PathType.Sum, Type.Min)
                .edgeWeight(x-> x.weight())
                .goalHasSolution(DistribuidorVertex.goalHasSolution())

```

```

        .heuristic(DistribuidorHeuristic::heuristic)
        .build();

    System.out.println("\n\n#### Ej3 Algoritmo PDR ####");
    PDR<DistribuidorVertex,DistribuidorEdge,?>    alg_pdr    =
PDR.of(graph);

    GraphPath<DistribuidorVertex,  DistribuidorEdge>    gp    =
alg_pdr.search().get();

    List<Integer> gp_as = gp.getEdgeList().stream()
        .map(x -> x.action())
        .collect(Collectors.toList());

    SolucionDistribuidor s_as = SolucionDistribuidor.of(gp_as);
    System.out.println(s_as);
}
}
}

```

```

> Resultados para el test 1

#### Ej3 Algoritmo PDR ####
Productos enviados a destinos:
Cantidad de producto0 en el destino[5, 0, 0, 0, 0]: 5
Cantidad de producto1 en el destino[1, 3, 8, 10, 5]: 27
Coste total: 67.0

> Resultados para el test 2
|

#### Ej3 Algoritmo PDR ####
Productos enviados a destinos:
Cantidad de producto0 en el destino[1, 3, 0, 0, 0, 0, 2]: 6
Cantidad de producto1 en el destino[0, 0, 2, 6, 4, 0, 0]: 12
Cantidad de producto2 en el destino[0, 0, 0, 0, 0, 8, 0]: 8
Coste total: 26.0

> Resultados para el test 3

#### Ej3 Algoritmo PDR ####
Productos enviados a destinos:
Cantidad de producto0 en el destino[0, 0, 0, 5, 0, 0, 0, 0, 2, 5]: 12
Cantidad de producto1 en el destino[0, 0, 0, 0, 1, 3, 1, 10, 0, 0]: 15
Cantidad de producto2 en el destino[3, 1, 8, 0, 0, 0, 0, 0, 0, 1]: 13
Coste total: 40.0

```

Test Manual

```
package Tests_Manuales;
```

```
import Datos.DatosDistribuidor;
```

```
import Ejercicio3_Manual.DistribuidorBT;
```

```
public class Ejercicio3_Test_Manual {
```

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

```
// TODO Auto-generated method stub
```

```
        System.out.println("##### Ejercicio 3 Manual#####");
```

```
        for (Integer id_fichero = 1; id_fichero <= 3; id_fichero++) {
```

```

        DatosDistribuidor.iniDatos("Ficheros/Ejercicio3DatosEntrada"
+ id_fichero + ".txt");

        System.out.println("\n\n>\tResultados para eltest " +
id_fichero + "\n");

        DistribuidorBT.search();

        System.out.println(DistribuidorBT.getSolucion() +
"\n");
    }
}
}
}

```

```

Ejercicios_Test_Manual.java Application] C:\Program Files\Java\jdk-20\bin\javaw.exe -v12 m...
##### Ejercicio 3 Manual#####

>      Resultados para eltest 1

Productos enviados a destinos:
Cantidad de producto0 en el destino[5, 0, 0, 0, 0]: 5
Cantidad de producto1 en el destino[1, 3, 8, 10, 5]: 27
Coste total: 67.0

>      Resultados para eltest 2

Productos enviados a destinos:
Cantidad de producto0 en el destino[1, 3, 0, 0, 0, 0, 2]: 6
Cantidad de producto1 en el destino[0, 0, 2, 6, 4, 0, 0]: 12
Cantidad de producto2 en el destino[0, 0, 0, 0, 0, 0, 8]: 8
Coste total: 26.0

```

Ejercicio 4

Datos

```
package Datos;
```

```
import java.util.List;
```

```
import java.util.Set;
```

```
import us.lsi.common.Files2;
```

```
import us.lsi.common.Set2;
```

```
public class DatosPersonas {
```

```
    public static List<Persona> personas;
```

```
    public record Persona(Integer codigo, Integer edad, Set<String>  
idiomas, String nacionalidad, Set<Afinidad> afinidades) {
```

```
        public static Persona create(String s) {
```

```
            String[] s1 = s.split(";");
```

```
            String[] s2 = s1[0].trim().split(" -> edad=");
```

```
            Integer codigo = Integer.valueOf(s2[0].trim().replace("P", ""));
```

```
            Integer edad = Integer.valueOf(s2[1].trim());
```

```
            String nacionalidad = s1[2].trim().replace("nacionalidad=", "");
```

```
            String idiomasStr = s1[1].trim().replace("idiomas=",  
"").replace("(", "").replace(")", "");
```

```
            Set<String> idiomas = Set2.parse(idiomasStr, ",",  
String::valueOf);
```

```
            String[] afinidadStr = s1[3].trim().split("afinidades=");
```

```
            Set<Afinidad> afinidades = Set2.parse(afinidadStr[1], ",",  
Afinidad::create);
```

```

        return new Persona(codigo, edad, idiomas, nacionalidad,
afinidades);
    }
}

```

```

public record Afinidad(Integer persona, Integer afinidad) {
    public static Afinidad create(String s) {
        String[] s1 = s.replace("(", "").replace(")", "").split(":");
        Integer persona = Integer.valueOf(s1[0].trim());
        Integer afinidad = Integer.valueOf(s1[1].trim());
        return new Afinidad(persona, afinidad);
    }
}

```

```

public static List<Persona> getPersonas() {
    return personas;
}

```

```

public static Integer getNumPersonas() {
    return personas.size();
}

```

```

public static Set<String> getIdiomas(Integer i) {
    return
        personas.stream().filter(p
p.codigo.equals(i)).findFirst().map(Persona::idiomas).get();
}

```

->

```

public static Integer getEdad(Integer i) {

```



```

        return      personas.stream().filter(p      ->
p.codigo.equals(i)).findFirst().map(Persona::edad).get();
    }

```

```

    public static String getNacionalidad(Integer i) {
        return      personas.stream().filter(p      ->
p.codigo.equals(i)).findFirst().map(Persona::nacionalidad).get();
    }

```

```

    public static Integer getAfinidad(Integer i, Integer j) {
        return      personas.stream().filter(p      ->
p.codigo.equals(i)).findFirst().map(Persona::afinidades).get().stream
().filter(p      ->
p.persona.equals(j)).findFirst().map(Afinidad::afinidad).get();
    }

```

```

    public static void iniDatos(String fichero) {
        personas      =      Files2.streamFromFile(fichero).filter(x->
x.contains("nacionalidad")).map(Persona::create).toList();
    }
}

```

Solucion

```
package Soluciones;
```

```
import java.util.ArrayList;
```

```
import java.util.List;
```

```
import java.util.Set;
```

```
import org.jgrapht.GraphPath;
```

```

import Datos.DatosPersonas;
import Ejercicio4.PersonaEdge;
import Ejercicio4.PersonaVertex;
import us.lsi.common.IntPair;

public record SolucionPersonas(Integer peso, List<IntPair> parejas)
implements Comparable<SolucionPersonas> {

    public static SolucionPersonas of(GraphPath<PersonaVertex,
PersonaEdge> path) {

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

        return SolucionPersonas.of(la);
    }

    public static SolucionPersonas of(List<Integer> value) {

        List<IntPair> p = new ArrayList<>();
        Integer afT = 0;
        Boolean b = true;
        Boolean b2 = true;
        Boolean b3 = true;
        List<Integer> cr = new ArrayList<>(value);
        if(cr.size() % 2 == 1) cr.remove(cr.size() - 1);
        for(int i = 0; i < cr.size() - 1; i+=2) {
            IntPair pareja = IntPair.of(cr.get(i), cr.get(i+1));
            p.add(pareja);
            afT += DatosPersonas.getAfinidad(cr.get(i), cr.get(i+1));
        }
    }
}

```

```

        for(IntPair par:p) {
            Set<String> intersection =
DatosPersonas.getIdiomas(par.first());

intersection.retainAll(DatosPersonas.getIdiomas(par.second()));

            b = b && !intersection.isEmpty();

            b2 = b2 && Math.abs(DatosPersonas.getEdad(par.first()) -
DatosPersonas.getEdad(par.second())) <= 5;

            b3 = b3 &&
!DatosPersonas.getNacionalidad(par.first()).equals(DatosPersonas.
getNacionalidad(par.second()));
        }
        System.out.println(b + " " + b2 + " " + b3);
        return new SolucionPersonas(aft, p);
    }

```

```

@Override
public String toString() {
    return String.format("Relacion de parejas: \n%s\nSuma de
afinidades: %d", parejas, peso);
}

```

```

@Override
public int compareTo(SolucionPersonas o) {
    return this.peso.compareTo(o.peso);
}
}

```

Edge

```
package Ejercicio4;
```

```
import Datos.DatosPersonas;
```

```
import us.lsi.graphs.virtual.SimpleEdgeAction;
```

```
public record PersonaEdge(PersonaVertex source, PersonaVertex  
target, Integer action, Double weight)
```

```
    implements SimpleEdgeAction<PersonaVertex, Integer>  
{
```

```
    public static PersonaEdge of(PersonaVertex v1, PersonaVertex  
v2, Integer action) {
```

```
        Double w = 0.;
```

```
        if (v1.index() % 2 == 1)
```

```
            w = DatosPersonas.getAfinidad(v1.ultima(), action)  
* 1.0;
```

```
        return new PersonaEdge(v1, v2, action, w);
```

```
    }
```

```
}
```

Vertex

```
package Ejercicio4;
```

```
import java.util.ArrayList;
```

```
import java.util.List;
```

```
import java.util.Set;
```

```
import java.util.function.Predicate;
```

```
import Datos.DatosPersonas;
```

```
import us.lsi.common.List2;
```

```
import us.lsi.common.Set2;
```

```
import us.lsi.graphs.virtual.VirtualVertex;
```

```
public record PersonaVertex(Integer index, Set<Integer> restante,  
Integer ultima) implements VirtualVertex<PersonaVertex,  
PersonaEdge, Integer> {
```

```
    public static PersonaVertex initial() {
```

```
        Set<Integer> restante = Set2.of();
```

```
        for(int i = 0; i < DatosPersonas.getNumPersonas(); i++) {
```

```
            restante.add(i);
```

```
        }
```

```
        return new PersonaVertex(0, restante,  
DatosPersonas.getNumPersonas());
```

```
    }
```

```
    public static Predicate<PersonaVertex> goal(){
```

```
        return v -> v.index() == DatosPersonas.getNumPersonas();
```

```
    }
```

```
    public static Predicate<PersonaVertex> goalHasSolution(){
```

```
        return v -> true;
```

```
    }
```

```
@Override
```

```
public List<Integer> actions() {
```

```
    List<Integer> alternativas = new ArrayList<>();
```

```
    if(index() == DatosPersonas.getNumPersonas()) {
```

```
        return List2.empty();
```

```

    } else if(index() % 2 == 1) {
        Set<String>          idiomasPersona          =
DatosPersonas.getIdiomas(this.ultima);

        Integer edadPersona = DatosPersonas.getEdad(this.ultima);

        String              nacionPersona           =
DatosPersonas.getNacionalidad(this.ultima);

        for(Integer pareja : this.restante) {

            Set<String>      idiomasPareja          =
DatosPersonas.getIdiomas(pareja);

            Integer edadPareja = DatosPersonas.getEdad(pareja);

            String           nacionPareja           =
DatosPersonas.getNacionalidad(pareja);

            Boolean          idiomaEnComun           =
idiomasPareja.stream().anyMatch(idioma            ->
idiomasPersona.contains(idioma));

            Integer  diferenciaEdad  =  Math.abs(edadPersona  -
edadPareja);

            Boolean  distintaNacionalidad           =
!nacionPersona.equals(nacionPareja);

            if(idiomaEnComun  &&  diferenciaEdad  <=  5  &&
distintaNacionalidad) {

                alternativas.add(pareja);

            }

        }

        return alternativas;

    } else {

        return List2.of(this.restante.stream().findFirst().get());

    }

}

```

@Override

```
public PersonaVertex neighbor(Integer a) {  
    Integer index = this.index + 1;  
    Set<Integer> res = Set2.copy(this.restante);  
    if(index % 2 == 0) {  
        Integer ult = DatosPersonas.getNumPersonas();  
        res.remove(a);  
        return new PersonaVertex(index, res, ult);  
    } else {  
        Integer ult = a;  
        res.remove(ult);  
        return new PersonaVertex(index, res, ult);  
    }  
}
```

@Override

```
public PersonaEdge edge(Integer a) {  
    return PersonaEdge.of(this, this.neighbor(a), a);  
}
```

```
public String toGraph() {  
    return this.index() + ", " + this.restante() + ", " + this.ultima();  
}
```

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

```
DatosPersonas.iniDatos("Ficheros/Ejercicio4DatosEntrada1.txt");  
    System.out.println(String.valueOf(initial()));
```

```

    }
}
Heuristic
package Ejercicio4;

import java.util.List;
import java.util.function.Predicate;
import java.util.stream.IntStream;

import Datos.DatosPersonas;

public class PersonaHeuristic {
    public static Double heuristic(PersonaVertex v1,
    Predicate<PersonaVertex> goal, PersonaVertex v2) {
        if(v1.ultima() == DatosPersonas.getNumPersonas())
            return 0.;
        else
            return IntStream.range(v1.ultima(),
    DatosPersonas.getNumPersonas())
                .mapToDouble(i -> mejorOpcion(i,
    v1.restante().stream().toList()))).sum();
    }

    private static Double mejorOpcion(Integer i, List<Integer>
    restante) {
        return restante.stream()
            .filter(j -> !j.equals(i))
            .mapToDouble(j -> DatosPersonas.getAfinidad(i, j)).max()
            .orElse(0);
    }
}

```



```
}
```

```
} package Ejercicio4;
```

```
import java.util.List;
```

```
import java.util.function.Predicate;
```

```
import java.util.stream.IntStream;
```

```
import Datos.DatosPersonas;
```

```
public class PersonaHeuristic {
```

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

```
        if(v1.ultima() == DatosPersonas.getNumPersonas())
```

```
            return 0.;
```

```
        else
```

```
            return IntStream.range(v1.ultima(),  
    DatosPersonas.getNumPersonas())
```

```
                .mapToDouble(i -> mejorOpcion(i,  
    v1.restante().stream().toList()))).sum();
```

```
    }
```

```
    private static Double mejorOpcion(Integer i, List<Integer>  
    restante) {
```

```
        return restante.stream()
```

```
            .filter(j -> !j.equals(i))
```

```
            .mapToDouble(j -> DatosPersonas.getAfinidad(i, j)).max()
```

```
            .orElse(0);
```

```
    }
```

}

BT

```
package Ejercicio4_Manual;

import Soluciones.SolucionPersonas;

public class PersonaBT {
    private static Double mejorValor;
    private static PersonaState state;
    private static SolucionPersonas solucion;

    public static void search() {
        solucion = null;
        mejorValor = Double.MIN_VALUE; // Estamos maximizando
        state = PersonaState.initial();
        bt_search();
    }

    private static void bt_search() {
        if (state.esTerminal()) {
            Double valorObtenido = state.acumulado;
            if (valorObtenido > mejorValor) { // Estamos maximizando
                mejorValor = valorObtenido;
                solucion = state.getSolucion();
            }
        } else {
            for (Integer a: state.actual.actions()) {
                Double cota = state.cota(a);
                if (cota < mejorValor) continue;
                state.forward(a);
                bt_search();
                state.back();
            }
        }
    }

    public static SolucionPersonas getSolucion() {
        return solucion;
    }
}
```

Problem

```
package Ejercicio4_Manual;
```

```
import java.util.ArrayList;
```

```
import java.util.List;
```

```
import java.util.Set;
```

```

import java.util.function.Predicate;
import java.util.stream.IntStream;

import Datos.DatosPersonas;
import us.lsi.common.List2;
import us.lsi.common.Set2;

public record PersonaProblem(Integer indice, Set<Integer> restante,
Integer ultima) {
    public static PersonaProblem initial() {
        Set<Integer> restante = Set2.of();
        for(int i = 0; i < DatosPersonas.getNumPersonas(); i++) {
            restante.add(i);
        }
        return new PersonaProblem(0, restante,
DatosPersonas.getNumPersonas());
    }

    public static Predicate<PersonaProblem> goal(){
        return obj -> obj.indice() == DatosPersonas.getNumPersonas();
    }

    public static Predicate<PersonaProblem> goalHasSolution(){
        return obj -> true;
    }

    public List<Integer> actions() {
        List<Integer> alternativas = new ArrayList<>();
    }

```

```

    if(indice() == DatosPersonas.getNumPersonas()) {
        return List2.empty();
    } else if(indice() % 2 == 1) {
        Set<String>          idiomasPersona          =
DatosPersonas.getIdiomas(this.ultima);

        Integer edadPersona = DatosPersonas.getEdad(this.ultima);

        String              nacionPersona           =
DatosPersonas.getNacionalidad(this.ultima);

        for(Integer pareja : this.restante) {
            Set<String>          idiomasPareja          =
DatosPersonas.getIdiomas(pareja);

            Integer edadPareja = DatosPersonas.getEdad(pareja);

            String              nacionPareja           =
DatosPersonas.getNacionalidad(pareja);

            Boolean             idiomaEnComun           =
idiomasPareja.stream().anyMatch(idioma
idiomasPersona.contains(idioma));

            Integer  diferenciaEdad  =  Math.abs(edadPersona  -
edadPareja);

            Boolean             distintaNacionalidad    =
!nacionPersona.equals(nacionPareja);

            if(idiomaEnComun  &&  diferenciaEdad  <=  5  &&
distintaNacionalidad) {
                alternativas.add(pareja);
            }
        }

        return alternativas;
    } else {
        return List2.of(this.restante.stream().findFirst().get());
    }
}

```

```
}
```

```
public PersonaProblem neighbor(Integer a) {  
    Integer index = this.indice + 1;  
    Set<Integer> res = Set2.copy(this.restante);  
    if(index % 2 == 0) {  
        Integer ult = DatosPersonas.getNumPersonas();  
        res.remove(a);  
        return new PersonaProblem(index, res, ult);  
    } else {  
        Integer ult = a;  
        res.remove(ult);  
        return new PersonaProblem(index, res, ult);  
    }  
}
```

```
public Double heuristic() {  
    if(this.ultima() == DatosPersonas.getNumPersonas())  
        return 0.;  
    else  
        return IntStream.range(this.ultima(),  
            DatosPersonas.getNumPersonas())  
            .mapToDouble(i -> mejorOpcion(i,  
this.restante().stream().toList()))).sum();  
}
```

```
private static Double mejorOpcion(Integer i, List<Integer> restante)  
{  
    return restante.stream()
```

```

        .filter(j -> j != i)
        .mapToDouble(j -> DatosPersonas.getAfinidad(i, j)).max()
        .orElse(0);
    }
}

```

State

```
package Ejercicio4_Manual;
```

```
import java.util.List;
```

```
import Datos.DatosPersonas;
```

```
import Soluciones.SolucionPersonas;
```

```
import us.lsi.common.List2;
```

```
public class PersonaState {
```

```
    PersonaProblem actual;
```

```
    Double acumulado;
```

```
    List<Integer> acciones;
```

```
    List<PersonaProblem> anteriores;
```

```
    private PersonaState(PersonaProblem p, Double a, List<Integer>
ls1, List<PersonaProblem> ls2) {
```

```
        actual = p;
```

```
        acumulado = a;
```

```
        acciones = ls1;
```

```
        anteriores = ls2;
```

```
    }
```

```

public static PersonaState initial() {
    PersonaProblem pi = PersonaProblem.initial();
    return of(pi, 0., List2.empty(), List2.empty());
}

```

```

public static PersonaState of(PersonaProblem prob, Double acum,
List<Integer> Isa, List<PersonaProblem> lsp) {
    return new PersonaState(prob, acum, Isa, lsp);
}

```

```

public void forward(Integer a) {
    if(actual.ultima() != DatosPersonas.getNumPersonas())
        acumulado += DatosPersonas.getAfinidad(actual.ultima(), a);
    acciones.add(a);
    anteriores.add(actual);
    actual = actual.neighbor(a);
}

```

```

public void back() {
    int last = acciones.size() - 1;
    var prob_ant = anteriores.get(last);
    if(prob_ant.ultima() != DatosPersonas.getNumPersonas())
        acumulado -= DatosPersonas.getAfinidad(prob_ant.ultima(),
acciones.get(last));
    acciones.remove(last);
    anteriores.remove(last);
    actual = prob_ant;
}

```

```
public List<Integer> alternativas() {  
    return actual.actions();  
}
```

```
public Double cota(Integer a) {  
    Integer weight = 0;  
    if(actual.ultima() != DatosPersonas.getNumPersonas())  
        weight = DatosPersonas.getAfinidad(actual.ultima(), a);  
    return acumulado + weight + actual.neighbor(a).heuristic();  
}
```

```
public Boolean esSolucion() {  
    return true;  
}
```

```
public boolean esTerminal() {  
    return PersonaProblem.goal().test(actual);  
}
```

```
public SolucionPersonas getSolucion() {  
    return SolucionPersonas.of(acciones);  
}  
}
```


Test A*

```
package Tests;
```

```
import java.util.List;
```

```
import java.util.Locale;
```

```
import org.jgrapht.GraphPath;
```

```
import Datos.DatosPersonas;
```

```
import Ejercicio4.PersonaEdge;
```

```
import Ejercicio4.PersonaHeuristic;
```

```
import Ejercicio4.PersonaVertex;
```

```
import Soluciones.SolucionPersonas;
```

```
import us.lsi.colors.GraphColors;
```

```
import us.lsi.colors.GraphColors.Color;
```

```
import us.lsi.graphs.alg.AStar;
```

```
import us.lsi.graphs.virtual.EDGraph;
```

```
import us.lsi.graphs.virtual.EDGraph.Type;
```

```
import us.lsi.path.EDGraphPath.PathType;
```

```
public class Ejercicio4Test {
```

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

```
        Locale.setDefault(Locale.of("en", "US"));
```

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

```
            DatosPersonas.iniDatos("Ficheros/Ejercicio4DatosEntrada" + i  
+ ".txt");
```

```
System.out.println("\n\n>\tResultados para el test " + i + "\n");
```

```
PersonaVertex start = PersonaVertex.initial();
```

```
EGraph<PersonaVertex, PersonaEdge> grafo =  
    EGraph.virtual(start, PersonaVertex.goal(),  
PathType.Sum, Type.Max)  
    .edgeWeight(x -> x.weight())  
  
    .goalHasSolution(PersonaVertex.goalHasSolution())  
    .heuristic(PersonaHeuristic::heuristic)  
    .build();
```

```
System.out.println("\n\n#### Ejercicio 4 Algoritmo A* ####");
```

```
AStar<PersonaVertex, PersonaEdge, ?> astar =  
    AStar.of(grafo);
```

```
GraphPath<PersonaVertex, PersonaEdge> camino =  
    astar.search().get();
```

```
List<Integer> camino_as =  
    camino.getEdgeList().stream()  
    .map(x -> x.action())  
    .toList();
```

```
SolucionPersonas sol = SolucionPersonas.of(camino_as);  
System.out.println(sol);
```

```

        GraphColors.toDot(astar.outGraph(),
            "Grafos_Generados/Ejercicio4/Ejercicio4Auto" + i +
".gv",
            v -> v.toGraph(),
            e -> e.action().toString() + ", " + e.weight().toString(),
            v      ->      GraphColors.colorIf(Color.green,
PersonaVertex.goal().test(v)),
            e      ->      GraphColors.colorIf(Color.green,
(camino.getEdgeList().contains(e))));
    }
}

}

```

```

>      Resultados para el test 1

#### Ejercicio 4 Algoritmo A* ####
true true true
Relacion de parejas:
[(0,2), (1,3), (4,5)]
Suma de afinidades: 15

>      Resultados para el test 2

#### Ejercicio 4 Algoritmo A* ####
true true true
Relacion de parejas:
[(0,5), (1,7), (2,4), (3,6)]
Suma de afinidades: 11

>      Resultados para el test 3

#### Ejercicio 4 Algoritmo A* ####
true true true
Relacion de parejas:
[(0,5), (1,3), (2,4), (6,7), (8,9)]
Suma de afinidades: 24

```

Test BT

```
package Tests;
```

```
import java.util.List;
```

```
import java.util.Locale;
```

```
import java.util.Optional;
```

```
import org.jgrapht.GraphPath;
```

```
import Datos.DatosPersonas;
import Ejercicio4.PersonaEdge;
import Ejercicio4.PersonaHeuristic;
import Ejercicio4.PersonaVertex;
import Soluciones.SolucionPersonas;
import us.lsi.colors.GraphColors;
import us.lsi.colors.GraphColors.Color;
import us.lsi.graphs.alg.BT;
import us.lsi.graphs.alg.GreedyOnGraph;
import us.lsi.graphs.virtual.EDGraph;
import us.lsi.graphs.virtual.EDGraph.Type;
import us.lsi.path.EDGraphPath.PathType;

public class Ejercicio4TestBT {

    public static void main(String[] args) {

        Locale.setDefault(Locale.of("en", "US"));

        for (int i = 1; i <= 3; i++) {
            DatosPersonas.iniDatos("Ficheros/Ejercicio4DatosEntrada"
+ i + ".txt");
            System.out.println("\n\n>\tResultados para el test " + i + "\n");

            PersonaVertex start = PersonaVertex.initial();

            EDGraph<PersonaVertex, PersonaEdge> grafo =
```

```

        EGraph.virtual(start, PersonaVertex.goal(),
PathType.Sum, Type.Max)
        .edgeWeight(x -> x.weight())
        .goalHasSolution(PersonaVertex.goalHasSolution())
        .heuristic(PersonaHeuristic::heuristic)
        .build();

```

```

System.out.println("\n\n#### Ejercicio 1 Algoritmo BT ####");

```

```

Boolean conVoraz = false;

```

```

SolucionPersonas sv = null;

```

```

Optional<GraphPath<PersonaVertex, PersonaEdge>> gp =
Optional.empty();

```

```

if (conVoraz) {
    GreedyOnGraph<PersonaVertex, PersonaEdge> ga =
GreedyOnGraph.of(grafo);
    gp = ga.search();
    if (gp.isPresent()) sv = SolucionPersonas.of(gp.get());
    System.out.println("Sv = "+sv);
}

```

```

BT<PersonaVertex, PersonaEdge, SolucionPersonas> bta =
null;

```

```

if(gp.isPresent())
    bta = BT.of(grafo, SolucionPersonas::of,
gp.get().getWeight(), gp.get(), true);

```

```

else
    bta = BT.of(grafo, null, null, null, true);

    bta.search();
    sv = SolucionPersonas.of(bta.optimalPath().orElse(null));

    List<PersonaEdge> le =
    bta.optimalPath().get().getEdgeList();

    System.out.println("Sol opt = "+sv);

    var outGraph = bta.outGraph();

    if(outGraph!=null)
        GraphColors.toDot(bta.outGraph(),
            "Grafos_Generados/Ejercicio4/Ejercicio4BT"+i+".gv",
            v -> v.toGraph(),
            e -> e.action().toString(),
            v -> GraphColors.colorIf(Color.red,
PersonaVertex.goal().test(v)),
            e -> GraphColors.colorIf(Color.red, le.contains(e)));

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

```

```
#### Ejercicio 1 Algoritmo BT ####
true true true
Sol opt = Relacion de parejas:
[(0,2), (1,3), (4,5)]
Suma de afinidades: 15
```

```
> Resultados para el test 2
```

```
#### Ejercicio 1 Algoritmo BT ####
true true true
Sol opt = Relacion de parejas:
[(0,5), (1,7), (2,4), (3,6)]
Suma de afinidades: 11
```

```
> Resultados para el test 3
```

```
#### Ejercicio 1 Algoritmo BT ####
true true true
Sol opt = Relacion de parejas:
[(0,5), (1,3), (2,4), (6,7), (8,9)]
Suma de afinidades: 24
```

Test PDR

```
package Tests;
```

```
import java.util.Locale;
```

```
import java.util.Optional;
```

```
import java.util.function.Predicate;
```



```

import org.jgrapht.GraphPath;

import Datos.DatosPersonas;
import Ejercicio4.PersonaEdge;
import Ejercicio4.PersonaHeuristic;
import Ejercicio4.PersonaVertex;
import Soluciones.SolucionPersonas;
import us.lsi.colors.GraphColors;
import us.lsi.colors.GraphColors.Color;
import us.lsi.graphs.alg.GreedyOnGraph;
import us.lsi.graphs.alg.PDR;
import us.lsi.graphs.virtual.EDGraph;
import us.lsi.graphs.virtual.EDGraph.Type;
import us.lsi.path.EDGraphPath.PathType;

public class Ejercicio4TestPDR {
    public static void main(String[] args) {
        Locale.setDefault(Locale.of("en", "US"));

        for (Integer id_fichero = 1; id_fichero <= 3; id_fichero++) {

            DatosPersonas.iniDatos("Ficheros/Ejercicio4DatosEntrada"+id_fichero+".txt");

            PersonaVertex vInicial = PersonaVertex.initial();
            Predicate<PersonaVertex> goal = PersonaVertex.goal();

            EDGraph<PersonaVertex, PersonaEdge> graph =

```

```

EGraph.virtual(vInicial, goal, PathType.Sum, Type.Max)
.goalHasSolution(PersonaVertex.goalHasSolution())
.heuristic(PersonaHeuristic::heuristic)
.build();

```

```

GreedyOnGraph<PersonaVertex, PersonaEdge> alg_voraz =
GreedyOnGraph.of(graph);

```

```

GraphPath<PersonaVertex, PersonaEdge> path =
alg_voraz.path();
path = alg_voraz.isSolution(path)? path: null;

```

```

PDR<PersonaVertex,PersonaEdge,SolucionPersonas>
alg_pdr = path==null?

```

```

PDR.of(graph):

```

```

PDR.of(graph, null, path.getWeight(), path, true);

```

```

Optional<GraphPath<PersonaVertex, PersonaEdge>> gp =
alg_pdr.search();

```

```

var res = alg_pdr.search().orElse(null);

```

```

var outGraph = alg_pdr.outGraph();

```

```

if(outGraph!=null) {

```

```

    GraphColors.toDot(alg_pdr.outGraph,

```

```

"Grafos_Generados/Ejercicio4/Ejercicio4PDR"+id_fichero+".gv",

```

```

    v -> v.toGraph(),

```

```

    e -> e.action().toString(),

```

```

    v -> GraphColors.colorIf(Color.red,
PersonaVertex.goal().test(v)),

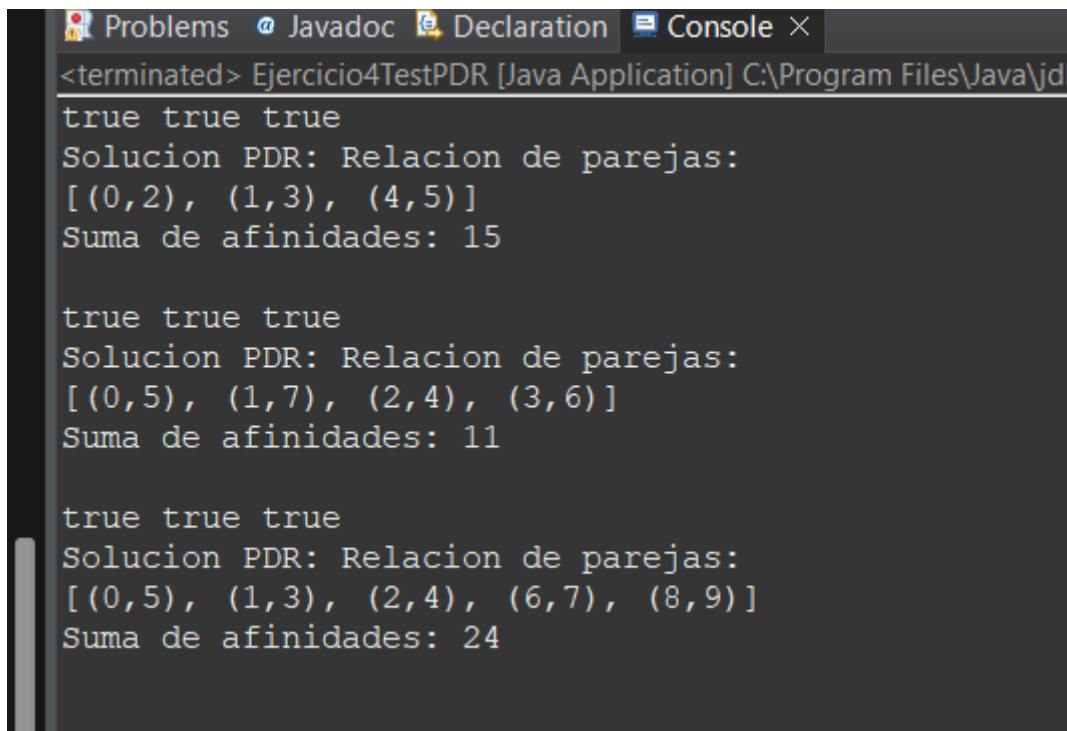
```

```

                e                ->                GraphColors.colorIf(Color.red,
gp.isPresent()?gp.get().getEdgeList().contains(e):false));
        }

        if(res!=null)
            System.out.println("Solucion PDR: " +
SolucionPersonas.of(res) + "\n");
        else
            System.out.println("PDR no obtuvo solucion\n");
    }
}
}
}

```



```

<terminated> Ejercicio4TestPDR [Java Application] C:\Program Files\Java\jdk
true true true
Solucion PDR: Relacion de parejas:
[(0,2), (1,3), (4,5)]
Suma de afinidades: 15

true true true
Solucion PDR: Relacion de parejas:
[(0,5), (1,7), (2,4), (3,6)]
Suma de afinidades: 11

true true true
Solucion PDR: Relacion de parejas:
[(0,5), (1,3), (2,4), (6,7), (8,9)]
Suma de afinidades: 24

```

Test Manual

```
package Tests_Manuales;
```

```
import Datos.DatosPersonas;
```

```
import Ejercicio4_Manual.PersonaBT;
```

```

public class Ejercicio4_Test_Manual {
    public static void main(String[] args) {
// TODO Auto-generated method stub

        System.out.println("##### Ejercicio 4
Manual#####");

        for (Integer id_fichero = 1; id_fichero <= 3; id_fichero++) {

            DatosPersonas.iniDatos("Ficheros/Ejercicio4DatosEntrada" +
id_fichero + ".txt");

                System.out.println("\n\n>\tResultados para eltest " +
id_fichero + "\n");

                PersonaBT.search();

                System.out.println(PersonaBT.getSolucion() + "\n");

            }

        }

    }
}

```

```
<terminated> Ejercicio4_Test_Manual [Java Application] C:\Program Files\Java\jdk-20\bin\ja
##### Ejercicio 4 Manual#####
```

```
> Resultados para eltest 1
```

```
true true true
Relacion de parejas:
[(0,2), (1,3), (4,5)]
Suma de afinidades: 15
```

```
> Resultados para eltest 2
```

```
true true true
Relacion de parejas:
[(0,5), (1,7), (2,4), (3,6)]
Suma de afinidades: 11
```

```
> Resultados para eltest 3
```

```
true true true
Relacion de parejas:
[(0,5), (1,3), (2,4), (6,7), (8,9)]
Suma de afinidades: 24
```