MEMORIA PRACTICA 5 ADDA

EJERCICIO1

DATOS EJERCICIO 1

```
package _datos;

import java.util.HashMap;
import java.util.List;
import java.util.Map;
import us.lsi.common.Files2;
import us.lsi.common.List2;

public class DatosEjercicio1 {
     public static record Tipo(Integer id, String nombre,
Integer kg) {

          public static int contt;

          public static Tipo create(String linea) {
               String[] v = linea.split(":");
               String nombre = v[0].trim();

               String[] v2 = v[1].split("=");
               Integer kg = Integer.parseInt
               (v2[1].trim().replace(";",""));

               return new Tipo(contt++, nombre, kg);
          }

          @Override
          public String toString() {
               return String.format

               ("%s: %d kgs", nombre, kg);

          }
     }

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

          public static int contv;

          public static Variedad create(String linea) {
               String[] v1 = linea.split(";");

               String[] v2 = v1[0].split("->"); // nombre :
beneficio
```

```
                String nombre = v2[0].trim();
                String[] v3 = v2[1].split("="); // beneficio
= num

                Integer beneficio =
Integer.parseInt(v3[1].trim());
                String[] v4 = v1[1].split("="); // comp =
tuplas
                String[] v5 = v4[1].split(","); // tuplas

                Map<String, Double> comp = new HashMap<>();
                for (String tupla : v5) {
                     tupla = tupla.trim().replace("(",
"").replace(")", "");
                     String[] v = tupla.split(":");
                     String nombreTipo = v[0].trim();
                     Double porcentaje =
Double.parseDouble(v[1].trim().replace(";", ""));
                     comp.put(nombreTipo, porcentaje);
                }
                return new Variedad(contv++, nombre,
beneficio, comp);
           }

           public String toString() {
                return String.format("%s", nombre);
           }
     }

     public static List<Tipo> tipos;
     public static List<Variedad> variedades;

     public static void iniDatos(String fichero) {
          Tipo.contt = 0;
          Variedad.contv = 0;
          tipos = List2.empty();
          variedades = List2.empty();
          List<String> lineas =
Files2.linesFromFile(fichero);
          Integer i2 = lineas.indexOf("// VARIEDADES");
          for (Integer i = 0; i < lineas.size(); i++) {
                if (i != 0 && i < i2) {
                     Tipo t = Tipo.create(lineas.get(i));
                     tipos.add(t);
                } else if (i > i2) {
                     Variedad v =
Variedad.create(lineas.get(i));
                     variedades.add(v);
                }
```

```
        }
    }

    public static Variedad getVariedad(Integer i) {

        return variedades.get(i);

    }

    public static Tipo getTipo(Integer j) {

        return tipos.get(j);

    }

    public static Tipo getTipo(String nombre) {
        return tipos.stream().filter(t ->

        t.nombre.equals(nombre)).findFirst().get();
    }

    public static Integer getKgTipo(Integer j) {

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

    }

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

    public static Map<String, Double>
getComponentesVariedad(Integer i) {
        return variedades.get(i).comp();
    }

    public static Double
getPorcentajeTipoVariedad(Integer i, Integer j) {
        Map<String, Double> comp =
getComponentesVariedad(i);
        String nombreT = tipos.get(j).nombre();
        return variedadContieneTipo(i,j) == 1 ?
comp.get(nombreT) : 0.0;
    }

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

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

	public static Integer variedadContieneTipo(Integer i,
Integer j) {

		Map<String, Double> comp =
getComponentesVariedad(i);
		String nombreT = tipos.get(j).nombre();
		return comp.keySet().contains(nombreT) ? 1 : 0;
	}

	// Test lectura de ficheros
	public static void main(String[] args) {
		iniDatos("ficheros/Ejercicio1DatosEntrada1.txt");
		System.out.println(tipos);
		System.out.println(variedades);
	}
}
```

SOLUCION EJERCICIO 1

```
package _soluciones;

import java.util.List;
import java.util.Map;
import java.util.stream.Collectors;
import org.jgrapht.GraphPath;
import _datos.DatosEjercicio1;
import _datos.DatosEjercicio1.Variedad;
import ejercicios.ejercicio1.Ejercicio1Vertex;
import ejercicios.ejercicio1.Ejercicio1Edge;
import us.lsi.common.Map2;

public class SolucionEjercicio1 {
	public static SolucionEjercicio1 of(List<Integer> ls) {
		return new SolucionEjercicio1(ls);
	}

	public static SolucionEjercicio1 of(GraphPath<Ejercicio1Vertex, Ejercicio1Edge>
path) {

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

		e.action()).toList();
```

```
		SolucionEjercicio1 res = of(ls);

		res.path = ls;
		return res;

	}

	private Integer beneficioTotal;

	private Map<Variedad, Integer> kilosVariedad;

	private List<Integer> path;

	private SolucionEjercicio1(List<Integer> ls) {
		beneficioTotal = 0;
		kilosVariedad = Map2.empty();
		for (int i = 0; i < ls.size(); i++) {
			if (ls.get(i) > 0) {
				Variedad v = DatosEjercicio1.getVariedad(i);
				Integer kg = ls.get(i);
				beneficioTotal += v.beneficio() * kg;
				kilosVariedad.put(v, kg);
			}
		}
	}

	@Override
	public String toString() {
		String intro = "Variedades de café seleccionadas:\n";
		String var = kilosVariedad.entrySet().stream().map(e -> e.getKey()+":
"+e.getValue() + " kgs")
		.collect(Collectors.joining("\n"));
		String ben ="\nBeneficio: " + beneficioTotal;
		String res = intro + var + ben +"\n";
		return path==null? res: String.format("%s\nPath de la solucion: %s", res,
path);
	}
}
```

EJERCICIO 1 EDGE

```
package ejercicios.ejercicio1;

import _datos.DatosEjercicio1;
import us.lsi.graphs.virtual.SimpleEdgeAction;
```

```
public record Ejercicio1Edge(Ejercicio1Vertex source,
Ejercicio1Vertex target, Integer action, Double weight)
     implements SimpleEdgeAction<Ejercicio1Vertex,
Integer> {


// Peso: beneficio que reporta
     public static Ejercicio1Edge of(Ejercicio1Vertex s,
Ejercicio1Vertex t, Integer a) {
          Double w = 0.;
          Integer indice = s.index();
          w = a *
DatosEjercicio1.getBeneficioVariedad(indice).doubleValue();

          return new Ejercicio1Edge(s, t, a, w);

     }
}


EJERCICIO 1 HEURISTIC
package ejercicios.ejercicio1;

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

import _datos.DatosEjercicio1;

public class Ejercicio1Heuristic {
     public static Double heuristic(Ejercicio1Vertex v1,
Predicate<Ejercicio1Vertex> goal, Ejercicio1Vertex v2) {

          return IntStream.range(v1.index(),
                    DatosEjercicio1.getNumVariedadesM())
                    .mapToDouble(variedad->
mejorOpcion(variedad,v1.remaining()))
                    .sum();
     }

     private static Double mejorOpcion(Integer variedad,
List<Integer> remaining) {
          Ejercicio1Vertex
          v = Ejercicio1Vertex.of(variedad, remaining);

          return IntStream.range(0,
Ejercicio1Vertex.maximaCantidadPosibleVariedad(v) + 1)
                    .filter(cant->
DatosEjercicio1.tipos.stream().allMatch(t -> cant *
```

```
     DatosEjercicio1.getPorcentajeTipoVariedad(v.index(),
t.id()) <= v.remaining().get(t.id())))
                    .boxed().mapToDouble(cant-> cant *

     DatosEjercicio1.getBeneficioVariedad(v.index()))
                    .max().orElse(-1000.);

     }
}
```

EJERCICIO 1 VERTEX

```
package ejercicios.ejercicio1;

import java.util.Comparator;
import java.util.List;
import java.util.Map;
import java.util.Map.Entry;
import java.util.function.Predicate;
import java.util.stream.Collectors;
import java.util.stream.IntStream;
import _datos.DatosEjercicio1;
import us.lsi.common.List2;
import us.lsi.graphs.virtual.VirtualVertex;

public record Ejercicio1Vertex(Integer index,
List<Integer> remaining)
          implements VirtualVertex<Ejercicio1Vertex,
Ejercicio1Edge, Integer> {

     public static Ejercicio1Vertex of(Integer i,
List<Integer> r) {

          return new Ejercicio1Vertex(i, r);

     }

     public static Ejercicio1Vertex initial() {
          return of(0, DatosEjercicio1.tipos.stream().map(t
->

          t.kg()).toList());
     }

     public static Predicate<Ejercicio1Vertex> goal() {
          return v -> v.index() ==
DatosEjercicio1.getNumVariedadesM();

     }
```

```
     public static Predicate<Ejercicio1Vertex>
goalHasSolution() {
          return v ->
v.remaining().stream().allMatch(cantidad

          -> cantidad

                    == 0);
     }

     public static Integer
maximaCantidadPosibleVariedad(Ejercicio1Vertex v) {

          Integer sol = Integer.MAX_VALUE;
          Map<String, Double> componentes =
DatosEjercicio1.getComponentesVariedad(v.index());

          for (Entry<String, Double> entrada :
componentes.entrySet()) {
               Integer tipo =
DatosEjercicio1.getTipo(entrada.getKey()).id();
               Double porcentaje = entrada.getValue();
               Integer restante = v.remaining().get(tipo);
               if (restante == 0) {
                    sol = 0;
                    break;
               } else {
                    Double cantidad = restante /
porcentaje;
                    if (cantidad < sol) {
                         sol = cantidad.intValue();
                    }
               }
          }
          return sol;

     }

     @Override
     public List<Integer> actions() {
          List<Integer> alternativas = List2.empty();
          if (goal().test(this)) {
               return alternativas;
          }
          Integer maximo =
maximaCantidadPosibleVariedad(this);
          for (Integer opcion = 0; opcion <= maximo;
opcion++) {
               alternativas.add(opcion);
```

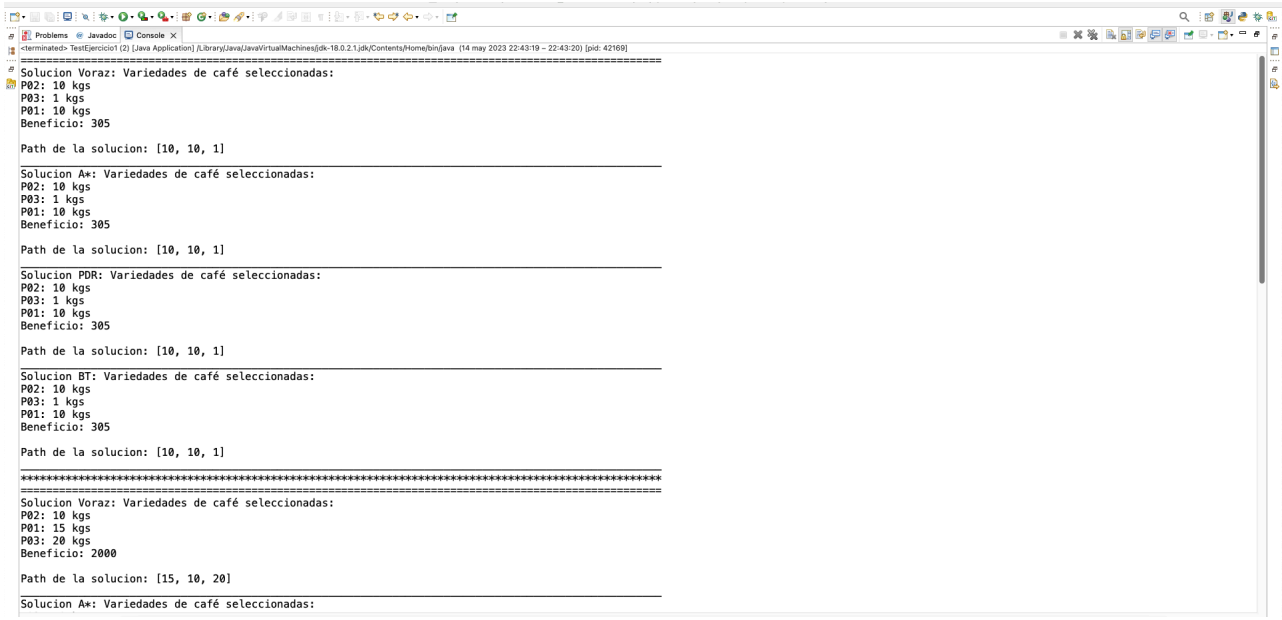
```
        }

        return alternativas;
    }

    @Override
    public Ejercicio1Vertex neighbor(Integer a) {
        Integer n_indice = this.index() + 1;
        List<Integer> n_remaining = List2.copy

        (this.remaining());

        if (a != 0) {
            Map<String, Double> porcentajes =


    DatosEjercicio1.getComponentesVariedad(this.index);
            Map<Integer, Double> cantidades =

                    porcentajes.entrySet().stream()


    .collect(Collectors.toMap(

                                            e ->


    DatosEjercicio1.getTipo(e.getKey()).id(), e ->
e.getValue() * a)

                                    );

            for (Entry<Integer, Double> entrada :
cantidades.entrySet())

            {

                n_remaining.set(

                            entrada.getKey(),
n_remaining.get(entrada.getKey()) -


    entrada.getValue().intValue()

                );

            }
        }
        return of(n_indice, n_remaining);
```

```
     }

     @Override
     public Ejercicio1Edge edge(Integer a) {
          return Ejercicio1Edge.of

          (this, this.neighbor(a), a);

     }

     // COMPROBAR
     public Ejercicio1Edge greedyEdge() {
          Comparator<Integer> cmp =
Comparator.comparing(cant -> cant *


     DatosEjercicio1.getBeneficioVariedad(index));

          Integer a = IntStream.range(0,

                    maximaCantidadPosibleVariedad

                    (this) + 1).filter(cant

          -> DatosEjercicio1.tipos.stream().allMatch(t ->
cant *


     DatosEjercicio1.getPorcentajeTipoVariedad(index,
t.id()) <=

                    remaining.get(t.id())))

                    .boxed().max(cmp).orElse(0);

          return edge(a);

     }
}
```

EJERCICIO 1 TEST

```
package ejercicios.tests;

import java.util.List;

import _datos.DatosEjercicio1;
```

```java
import _soluciones.SolucionEjercicio1;
import _utils.GraphsPI5;
import _utils.TestsPI5;
import ejercicios.ejercicio1.Ejercicio1Vertex;

public class TestEjercicio1 {

	public static void main(String[] args) {
		List.of(1,2,3).forEach(num_test -> {
			TestsPI5.iniTest("Ejercicio1DatosEntrada",
num_test, DatosEjercicio1::iniDatos);

			// TODO Defina en el tipo vertice un m.
factoria para el vertice inicial
			// TODO Defina en el tipo vertice un m.
static / Predicate para los vertices finales
			TestsPI5.tests(
				Ejercicio1Vertex.initial(),		//
Vertice Inicial
				Ejercicio1Vertex.goal(),		//
Predicado para un vertice final
				GraphsPI5::ejercicio1Builder,
	// Referencia al Builder del grafo
				Ejercicio1Vertex::greedyEdge,//
Referencia a la Funcion para la arista voraz
				SolucionEjercicio1::of);		//
Referencia al metodo factoria para la solucion
		});
	}

}
```



EJERCICIO 2

DATOS CURSOS

```
package _datos;

import java.util.ArrayList;
import java.util.HashSet;
import java.util.List;
import java.util.Set;
import us.lsi.common.Files2;

public class DatosCursos {

	public static List<Curso> cursos;
	public static Set<Integer> tematicas;
	public static Integer maxCentros;

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

		public static int cont;
		public static Curso of(Set<Integer> temsCurso,
Double prec, Integer centr) {
			return new Curso(cont++, temsCurso, prec,
centr);
		}
	}

	public static void iniDatos(String fichero) {
		tematicas = new HashSet<>();
		cursos = new ArrayList<>();
		maxCentros = 0;

		Files2.linesFromFile(fichero).forEach(l -> {

			if (l.startsWith("M")) {
				String[] c = l.split("=");
				maxCentros =
Integer.valueOf(c[1].trim());
			} else {
				String[] c = l.split(":");
				Set<Integer> temsCurso = new
HashSet<>();
				String[] tems = c[0].replace("{",
"").replace("}", "").split(",");
				for (String tem : tems) {
					Integer tematica =
Integer.valueOf(tem);
					temsCurso.add(tematica);
```

```
                        tematicas.add(tematica);
                    }
                    Double coste =
Double.valueOf(c[1].trim());
                    Integer centro =
Integer.valueOf(c[2].trim());
                    Curso curs = Curso.of(temsCurso, coste,
centro);
                    cursos.add(curs);
                }
            });

    }

    public static Integer getMaxCentros() {
        return maxCentros;
    }

    public static Integer getCentroCurso(Integer i) {
        return getCurso(i).centro();
    }

    public List<Curso> cursosImpartenTematica(Integer i)
{
        List<Curso> res = new ArrayList<>();
        for (Curso c : cursos) {
            if (c.tematicas().contains(i)) {
                res.add(c);
            }
        }
        return res;
    }

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

    public static Integer getNumCursos() {

        return getCursos().size();
    }

    public static Double getPrecioCurso(Integer i) {

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

    public static Curso getCurso(Integer i) {
        return cursos.get(i);
    }
```

```
	public static Set<Integer> getTematicas() {
		return tematicas;
	}

	public static Integer getNumTematicas() {
		return getTematicas().size();
	}

	public static Set<Integer> getTematicasCurso(Integer
i) {

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

	public static void toConsole(){
		System.out.println( "Maximo numero de colegios a
elegir: "+maxCentros+ "\nCursos disponibles: "+cursos);
		}

	public static void main(String[] args){
		System.out.println("DATOS DE ENTRADA 1:");
		iniDatos("ficheros/Ejercicio2DatosEntrada1.txt");

System.out.println("_____________________________");
		System.out.println("DATOS DE ENTRADA 2:");
		iniDatos("ficheros/Ejercicio2DatosEntrada2.txt");

System.out.println("_____________________________");
		System.out.println("DATOS DE ENTRADA 3:");
		iniDatos("ficheros/Ejercicio2DatosEntrada3.txt");
		}
}
```

SOLUCION CURSOS

```
package _soluciones;

import java.util.ArrayList;
import java.util.List;
import java.util.stream.Collectors;
import org.jgrapht.GraphPath;
import _datos.DatosCursos;
import _datos.DatosCursos.Curso;
import ejercicios.ejercicio2.CursoEdge;
import ejercicios.ejercicio2.CursoVertex;
```

```java
public class SolucionCursos implements
Comparable<SolucionCursos> {

     public static SolucionCursos of_Range(List<Integer>
ls) {
          return new SolucionCursos(ls);
     } // Ahora en la PI5

     public static SolucionCursos
of(GraphPath<CursoVertex, CursoEdge> path) {
          List<Integer> ls =
path.getEdgeList().stream().map(e -> e.action()).toList();
          SolucionCursos res = of_Range(ls);
          res.path = ls;
          return res;
     }

     private Double precio;
     private List<Curso> cursos;

     private List<Integer> path;

     public SolucionCursos() {
          precio = 0.;
          cursos = new ArrayList<>();
     }

     public SolucionCursos(List<Integer> ls) {
          precio = 0.;
          cursos = new ArrayList<>();

          for (int i = 0; i < ls.size(); i++) {
               if (ls.get(i) > 0) {
                    precio +=
DatosCursos.getPrecioCurso(i);
                    cursos.add(DatosCursos.cursos.get(i));
               }
          }
     }

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

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

```
          String res = String.format("%sCoste Total: %.1f",
s, precio);
          return path == null ? res :
String.format("%s\nPath de la solucion: %s", res, path);
     }

     @Override
     public int compareTo(SolucionCursos o) {
          return precio.compareTo(o.precio);
     }
}
```

CURSOS EDGE

```
package ejercicios.ejercicio2;

import _datos.DatosCursos;
import us.lsi.graphs.virtual.SimpleEdgeAction; //aqui no
tocamos acsi nada con respecto a los ejemplos

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

     public static CursoEdge of(CursoVertex s, CursoVertex
t, Integer a) {
          return new CursoEdge(s, t, a, a *
DatosCursos.getPrecioCurso(s.index()));
     }

     public String toString() {
          return String.format("%d; %.1f", action, weight);
     }
}
```

CURSOS HEURISTIC

```
package ejercicios.ejercicio2;

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

import _datos.DatosCursos;
import us.lsi.common.List2;

public class CursoHeuristic {
     public static Double heuristic(CursoVertex a,
Predicate<CursoVertex> goal, CursoVertex b) {
```

```
            return a.remaining().isEmpty() ? 0.
                        : IntStream.range(a.index(),
DatosCursos.getNumCursos())
                                    .filter(v ->
!List2.intersection(a.remaining(),
DatosCursos.getTematicasCurso(v)).isEmpty())
                                    .mapToDouble(v ->
DatosCursos.getPrecioCurso(v)).min().orElse(100.);
      }
}
```

CURSOS VERTEX

```
package ejercicios.ejercicio2;

import java.util.List;
import java.util.Set;
import java.util.function.Predicate;
import _datos.DatosCursos;
import us.lsi.common.List2;
import us.lsi.common.Set2;
import us.lsi.graphs.virtual.VirtualVertex;

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

        public static CursoVertex of(Integer i, Set<Integer> set, Set<Integer> centros) {
                return new CursoVertex(i, set, centros);
        }

        public static CursoVertex initial() {
                return of(0, Set2.copy(DatosCursos.getTematicas()), Set2.empty());
        }

        public static Predicate<CursoVertex> goal() {
                return v -> v.index() == DatosCursos.getNumCursos();
        }

        public static Predicate<CursoVertex> goalHasSolution() {
                return v -> v.remaining().isEmpty();
        }

        @Override
        public List<Integer> actions() {

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

                if (index < DatosCursos.getNumCursos()) {
                        if (remaining.isEmpty()) {
                                alternativas = List2.of(0);
                        } else {
                                Set<Integer> restantesActualizados = Set2.difference(remaining,
DatosCursos.getTematicasCurso(index));
                                if (index == DatosCursos.getNumCursos() - 1) {

                                        if (centros.contains(DatosCursos.getCentroCurso(index))
                                                        || (centros.size() < DatosCursos.maxCentros)) {
                                                alternativas = restantesActualizados.isEmpty() ? List2.of(1)
: List2.of(0);
                                        }
```

```
                                        } else if (restantesActualizados.equals(remaining)) {
                                                alternativas = List2.of(0);

                                        } else {

                                                if (centros.contains(DatosCursos.getCentroCurso(index))
                                                                || (centros.size() < DatosCursos.maxCentros)) {
                                                        alternativas = List2.of(0);
                                                        alternativas.add(1);
                                                } else {
                                                        alternativas = List2.of(0);
                                                }
                                        }
                                }
                        }
                        return alternativas;

        }

        @Override
        public CursoVertex neighbor(Integer a) {
                Set<Integer> rest1 = a == 0 ? Set2.copy(remaining)
                                : Set2.difference(remaining, DatosCursos.getTematicasCurso(index));
                Set<Integer> coles = Set2.copy(centros);
                if (a == 1) {
                        coles.add(DatosCursos.getCentroCurso(index));
                }

                return of(index + 1, rest1, coles);
        }

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

        public CursoEdge greedyEdge() {
                CursoEdge res = null;
                Set<Integer> restantesActualizados = Set2.difference(remaining,
DatosCursos.getTematicasCurso(index));

                if (centros.contains(DatosCursos.getCentroCurso(index)) || (centros.size() <
DatosCursos.maxCentros)) {
                        res = restantesActualizados.equals(remaining) ? edge(0) : edge(1);
                } else {
                        res = edge(0);
                }
                return res;
        }

        public String toString() {
                return String.format("%d; %d", index, remaining.size());
        }
}
```

TEST EJERCICIO 2

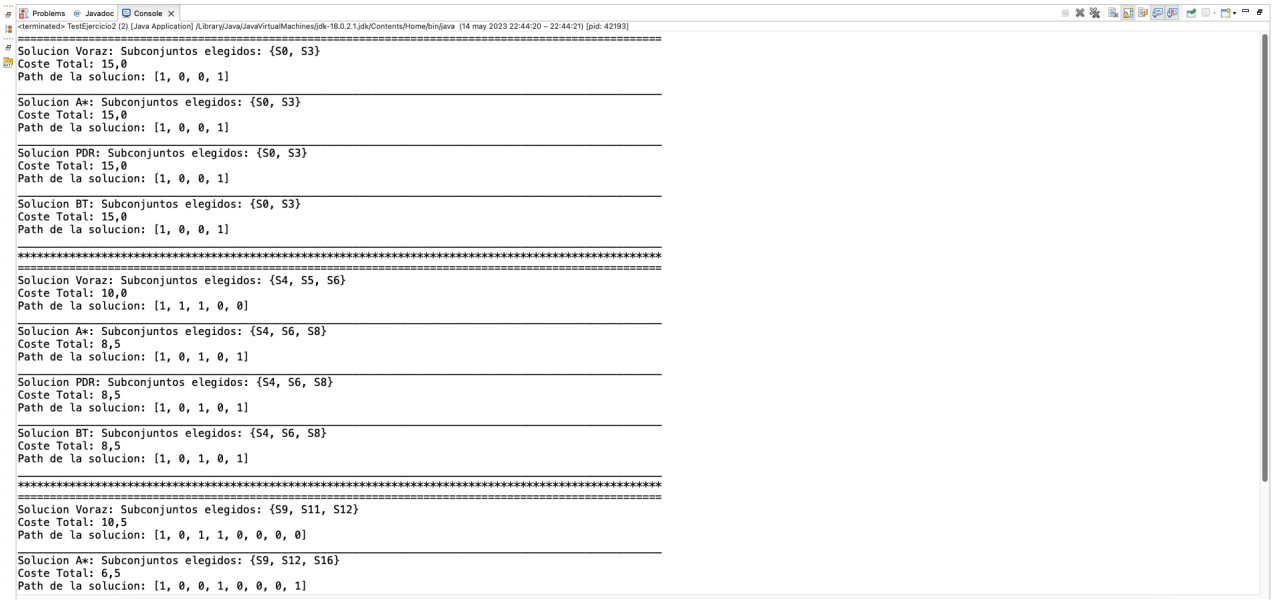
```
package ejercicios.tests;

import java.util.List;
```

```java
import _datos.DatosCursos;
import _soluciones.SolucionCursos;
import _utils.GraphsPI5;
import _utils.TestsPI5;
import ejercicios.ejercicio2.CursoVertex;

public class TestEjercicio2 {
     public static void main(String[] args) {
          List.of(1, 2, 3).forEach(num_test -> {
               TestsPI5.iniTest("Ejercicio2DatosEntrada",
num_test, DatosCursos::iniDatos);

               TestsPI5.tests(
                         CursoVertex.initial(), // Vertice
Inicial
                         CursoVertex.goal(), // Predicado
para un vertice final
                         GraphsPI5::cursosBuilder, //
Referencia al Builder del grafo
                         CursoVertex::greedyEdge, //
Referencia a la Funcion para la arista voraz
                         SolucionCursos::of); // Referencia
al metodo factoria para la solucion
          });
     }

}
```

TEST MANUAL EJ2

```java
package ejercicios.tests;

import java.util.List;
import _datos.DatosCursos;
import _utils.TestsPI5;
import ejercicios.ejercicio2.manual.CursosPDR;
import us.lsi.common.String2;

public class TestEjercicioM2 {

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


     DatosCursos.iniDatos("ficheros/Ejercicio2DatosEntrada"
+ num_test + ".txt");
               String2.toConsole("Solucion obtenida: %s\n",
CursosPDR.search());
               TestsPI5.line("*");
```

```
            });
        }
}
```



EJERCICIO 4

DATOS CLIENTES

```
package _datos;

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

import org.jgrapht.Graph;

import us.lsi.graphs.Graphs2;
import us.lsi.graphs.GraphsReader;

public class DatosClientes {

        public static Graph<Cliente, Carretera> grafo;
```

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

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

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

		}

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

	}

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

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

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

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

	public static Integer getNumVertices() {
		return grafo.vertexSet().size();
```

```
	}

	public static Set<Integer> getClientes() {
		return grafo.vertexSet().stream().map(x ->
x.id()).collect(Collectors.toSet());
	}

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

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

	public static Boolean existeArista(Integer i, Integer j) {
		Cliente c1 = getCliente(i);
		Cliente c2 = getCliente(j);
		return grafo.containsEdge(c1, c2);
	}

	public static Double getPeso(Integer i, Integer j) {
		Cliente c1 = getCliente(i);
		Cliente c2 = getCliente(j);
		return grafo.getEdge(c1, c2).distancia();

	}

	public static void main(String[] args) {
		initDatos("ficheros/Ejercicio4DatosEntrada1.txt");
		System.out.println(getPeso(2, 4));
	}

}
```

SOLUCION CLIENTES

```
package _soluciones;
```

```
import java.util.List;

import org.jgrapht.GraphPath;

import _datos.DatosClientes;
import ejercicios.ejercicio4.ClientesEdge;
import ejercicios.ejercicio4.ClientesVertex;


public class SolucionClientes implements Comparable<SolucionClientes> {

	public static SolucionClientes of_format(List<Integer> ls) {
		return new SolucionClientes(ls);
	}

	// Ahora en la PI5
	public static SolucionClientes of(GraphPath<ClientesVertex, ClientesEdge> path)
{
		List<Integer> ls = path.getEdgeList().stream().map(e ->
e.action()).toList();
		SolucionClientes res = of_format(ls);
		res.path = ls;
		return res;
	}

	private Double total;
	private Double kms;

	// Ahora en la PI5
	private List<Integer> path;

	private SolucionClientes(List<Integer> ls) {
		kms = DatosClientes.getPeso(0, ls.get(0));
		total = DatosClientes.getBeneficio(ls.get(0)) - kms;
		for (int i = 1; i < ls.size(); i++) {
			if (i == ls.size() - 1) {
				total += DatosClientes.getBeneficio(ls.get(i))
						- (kms + DatosClientes.getPeso(ls.get(i - 1),
ls.get(i)));
			} else {
				kms += DatosClientes.getPeso(ls.get(i - 1), ls.get(i));
				total += DatosClientes.getBeneficio(ls.get(i)) - kms;
			}
		}

	}
```

```
	// Ahora en la PI5
	@Override
	public String toString() {
		String res = String.format("Beneficio total:" + total + "\nKMs: " + kms);
		return path == null ? res : String.format("%s\nPath de la solucion
partiendo desde 0: %s", res, path);
	}

	@Override
	public int compareTo(SolucionClientes s) {
		return total.compareTo(s.total);
	}

}
```

CLIENTE HEURISTICA

```
package ejercicios.ejercicio4;

import java.util.function.Predicate;

public class ClienteHeuristica {

    public static Double
heuristic(ClientesVertex v1,
Predicate<ClientesVertex> goal,
ClientesVertex v2) {
        return 10000.;
    }

}
```

CLIENTES BT

```
package ejercicios.ejercicio4;

import java.util.Set;

import ejercicios.ejercicio4.*;
import _soluciones.SolucionClientes;
import us.lsi.common.Set2;
```

```
public class ClientesBT {

	private static Double mejorValor;
	private static ClientesEstado estado;
	private static Set<SolucionClientes> soluciones;

	public static void search() {
		soluciones = Set2.newTreeSet();
		mejorValor = Double.MIN_VALUE;
		estado = ClientesEstado.initial();
		bt_search();
	}

	private static void bt_search() {
		if (estado.esSolucion()) {
			Double valorObtenido = estado.acumulado;
			if (valorObtenido > mejorValor) {
				mejorValor = valorObtenido;
				soluciones.add(estado.getSolucion());
			}
		} else if (!estado.esTerminal()) {

			for (Integer a : estado.alternativas()) {
				if (estado.cota(a) >= mejorValor) {

					estado.forward(a);
					bt_search();
					estado.back();
				}
			}
		}
	}

	public static Set<SolucionClientes> getSoluciones() {
		return soluciones;
	}

}
```

CLIENTES EDGE

```
package ejercicios.ejercicio4;

import _datos.DatosClientes;
import us.lsi.graphs.virtual.SimpleEdgeAction;

public record ClientesEdge(ClientesVertex source, ClientesVertex target, Integer action,
Double weight)
```

```
		implements SimpleEdgeAction<ClientesVertex, Integer> {

	public static ClientesEdge of(ClientesVertex s, ClientesVertex t, Integer a) {
		// TODO La arista debe tener peso
		return new

		ClientesEdge(s, t, a, DatosClientes.getBeneficio(t.index()));
	}
}
```

CLIENTES ESTADO

```
package ejercicios.ejercicio4;

import java.util.List;

import _datos.DatosClientes;
import _soluciones.SolucionClientes;
import us.lsi.common.List2;
import ejercicios.ejercicio4.*;

public class ClientesEstado {

	ClientesProblem actual;
	Double acumulado;
	List<Integer> acciones;
	List<ClientesProblem> anteriores;

	private ClientesEstado(ClientesProblem p, Double a, List<Integer> ls1,
List<ClientesProblem> ls2) {
		actual = p;
		acumulado = a;
		acciones = ls1;
		anteriores = ls2;
	}

	public static ClientesEstado initial() {
		ClientesProblem p = ClientesProblem.initial();
		Double a = 0.;
		List<Integer> ls1 = List2.empty();
		List<ClientesProblem> ls2 = List2.empty();
		return new ClientesEstado(p, a, ls1, ls2);
	}

	public static ClientesEstado of(ClientesProblem prob, Double acum,
List<Integer> lsa, List<ClientesProblem> lsp) {
		return new ClientesEstado(prob, acum, lsa, lsp);
	}
```

```
	public void forward(Integer a) {
		acumulado += a * DatosClientes.getBeneficio(actual.index());
		acciones.add(a);
		anteriores.add(actual);
		actual = actual.neighbor(a);
	}

	public void back() {
		int last = acciones.size() - 1;
		ClientesProblem prob_ant = anteriores.get(last);
		acumulado = acciones.get(last) *
DatosClientes.getBeneficio(prob_ant.index());
		acciones.remove(last);
		anteriores.remove(last);
		actual = prob_ant;

	}

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

	public Double cota(Integer a) {
		Double weight = DatosClientes.getBeneficio(a);
		return acumulado + weight + actual.neighbor(a).heuristic();
	}

	public Boolean esSolucion() {

		return actual.index() == 0 && actual.pendientes().isEmpty();
	}

	public Boolean esTerminal() {
		return actual.index() == 0 && actual.pendientes().isEmpty();
	}

	public SolucionClientes getSolucion() {
		return SolucionClientes.of_format(acciones);
	}

}
```

CLIENTES PROBLEM

```
package ejercicios.ejercicio4;

import java.util.List;
```

```
import java.util.Set;
import java.util.function.Predicate;
import java.util.stream.Collectors;

import _datos.DatosClientes;
import us.lsi.common.List2;
import us.lsi.common.Set2;

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

	public static ClientesProblem of(Integer i, Set<Integer> pend, List<Integer>
visitados, Double kms) {
		return new ClientesProblem(i, pend, visitados, kms);
	}

	public static ClientesProblem initial() {
		return of(0, Set2.copy(DatosClientes.getClientes()), List2.of(0), 0.);
	}

	public static Predicate<ClientesVertex> goal() {
		return v -> v.index() == 0 && v.pendientes().isEmpty();

	}

	public Boolean existeCaminoDeVuelta(Integer accion) {
		Boolean res = true;
		List<Integer> restantes = pendientes.stream().collect(Collectors.toList());
		restantes.remove(accion);

		int i = 1;
		while (i < restantes.size()) {
			res = DatosClientes.existeArista(restantes.get(i), 0);
			if (res.equals(true)) {
				break;

			} else {
				res = false;
				i++;
			}
		}
		return res;
	}

	public List<Integer> actions() {
		List<Integer> alternativas = List2.empty();
```

```
                if (visitados.size() == DatosClientes.getNumVertices()) {
                        if (DatosClientes.existeArista(index, 0)) {
                                alternativas.add(0);
                        }
                } else {
                        for (Integer elem : pendientes) {
                                if (DatosClientes.existeArista(index, elem)) {
                                        if (existeCaminoDeVuelta(elem)) {
                                                alternativas.add(elem);
                                        }
                                }
                        }
                }
                return alternativas;
        }

        public ClientesProblem neighbor(Integer a) {
                Set<Integer> rest = Set2.copy(pendientes);
                rest.remove(a);
                List<Integer> vis = List2.copy(visitados);
                vis.add(a);

                return of(a, rest, vis, kms + DatosClientes.getPeso(index, a));
        }

        public Double heuristic() {
                return 10000.;
        }

}
```

CLIENTES VERTEX

```
package ejercicios.ejercicio4;

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

import _datos.DatosClientes;
import us.lsi.common.Set2;
import us.lsi.common.List2;
import us.lsi.graphs.virtual.VirtualVertex;

public record ClientesVertex(Integer index, Set<Integer> pendientes, List<Integer>
visitados, Double kms)
```

```
		implements VirtualVertex<ClientesVertex, ClientesEdge, Integer> {

	public static ClientesVertex of(Integer i, Set<Integer> pend, List<Integer>
visitados, Double kms) {
		return new ClientesVertex(i, pend, visitados, kms);
	}

	public static ClientesVertex initial() {
		return of(0, Set2.copy(DatosClientes.getClientes()), List2.of(0), 0.);
	}

	public static Predicate<ClientesVertex> goal() {
		return v -> v.index() == 0 && v.pendientes().isEmpty();

	}

	public static Predicate<ClientesVertex> goalHasSolution() {
		return v -> v.index() == 0 && v.pendientes().isEmpty();
	}

	// TODO Consulte las clases GraphsPI5 y TestPI5

	public Boolean existeCaminoDeVuelta(Integer accion) {
		Boolean res = true;
		List<Integer> restantes = pendientes.stream().collect(Collectors.toList());
		restantes.remove(accion);
		int i = 1;
		while (i < restantes.size()) {
			res = DatosClientes.existeArista(restantes.get(i), 0);
			if (res.equals(true)) {
				break;

			} else {
				res = false;
				i++;
			}
		}
		return res;
	}

	@Override
	public List<Integer> actions() {
		List<Integer> alternativas = List2.empty();

		if (visitados.size() == DatosClientes.getNumVertices()) {
			if (DatosClientes.existeArista(index, 0)) {
				alternativas.add(0);
```

```
                }
            } else {
                for (Integer elem : pendientes) {
                    if (DatosClientes.existeArista(index, elem)) {
                        if (existeCaminoDeVuelta(elem)) {
                            alternativas.add(elem);
                        }
                    }
                }
            }
            return alternativas;
        }

        @Override
        public ClientesVertex neighbor(Integer a) {
            Set<Integer> rest = Set2.copy(pendientes);
            rest.remove(a);
            List<Integer> vis = List2.copy(visitados);
            vis.add(a);
            return of(a, rest, vis, kms + DatosClientes.getPeso(index, a));
        }

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

        }

        // Se explica en practicas.
        public ClientesEdge greedyEdge() {
            return null;
        }

}
```

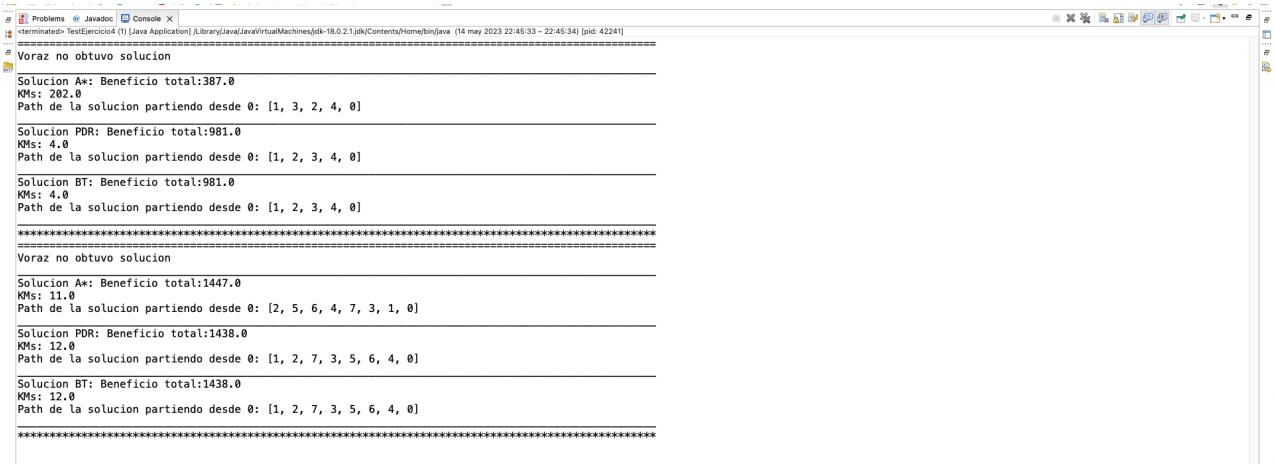
TEST EJERCICIO 4

```
package ejercicios.tests;

import java.util.List;
import _datos.DatosClientes;
import _soluciones.SolucionClientes;
import _utils.GraphsPI5;
import _utils.TestsPI5;
import ejercicios.ejercicio4.ClientesVertex;

public class TestEjercicio4 {
```

```
	public static void main(String[] args) {
		List.of(1,2).forEach(num_test -> {
			TestsPI5.iniTest("Ejercicio4DatosEntrada", num_test,
DatosClientes::initDatos);

			TestsPI5.tests(
					ClientesVertex.initial(),
					ClientesVertex.goal(),
					GraphsPI5::clientesBuilder,
					ClientesVertex::greedyEdge,
					SolucionClientes::of);
		});
	}
}
```

TEST MANUAL EJ4

```
package ejercicios.tests;

import java.util.List;

import _datos.DatosClientes;
import _utils.TestsPI5;
import ejercicios.ejercicio4.ClientesBT;
import us.lsi.common.String2;

public class TestEjercicioM4 {
	public static void main(String[] args) {
		List.of(1, 2).forEach(num_test -> {

			DatosClientes.initDatos("ficheros/Ejercicio4DatosEntrada" +
num_test + ".txt");
			ClientesBT.search();
			ClientesBT.getSoluciones().forEach(s ->
String2.toConsole("Solucion obtenida: %s\n", s));
			TestsPI5.line("*");
		});
	}

}
```


Problems
Javadoc
Console
<terminated> TestEjercicio4 (1) [Java Application] /Library/Java/JavaVirtualMachines/jdk-18.0.2.1.jdk/Contents/Home/bin/java (14 may 2023 22:45:33 – 22:45:34) [pid: 42241]
Voraz no obtuvo solucion
Solucion A*: Beneficio total:387.0
KMs: 202.0
Path de la solucion partiendo desde 0: [1, 3, 2, 4, 0]
Solucion PDR: Beneficio total:981.0
KMs: 4.0
Path de la solucion partiendo desde 0: [1, 2, 3, 4, 0]
Solucion BT: Beneficio total:981.0
KMs: 4.0
Path de la solucion partiendo desde 0: [1, 2, 3, 4, 0]
Voraz no obtuvo solucion
Solucion A*: Beneficio total:1447.0
KMs: 11.0
Path de la solucion partiendo desde 0: [2, 5, 6, 4, 7, 3, 1, 0]
Solucion PDR: Beneficio total:1438.0
KMs: 12.0
Path de la solucion partiendo desde 0: [1, 2, 7, 3, 5, 6, 4, 0]
Solucion BT: Beneficio total:1438.0
KMs: 12.0
Path de la solucion partiendo desde 0: [1, 2, 7, 3, 5, 6, 4, 0]


Problems
Javadoc
Console
<terminated> TestEjercicio4 [Java Application] /Library/Java/JavaVirtualMachines/jdk-18.0.2.1.jdk/Contents/Home/bin/java (14 may 2023 22:46:03 – 22:46:03) [pid: 42249]
Solucion obtenida: Beneficio total:-108.0
KMs: 301.0
Solucion obtenida: Beneficio total:387.0
KMs: 202.0
Solucion obtenida: Beneficio total:981.0
KMs: 4.0
Solucion obtenida: Beneficio total:1426.0
KMs: 14.0
Solucion obtenida: Beneficio total:1438.0
KMs: 12.0