

PRACTICA INDIVIDUAL 5

MEMORIA 5: Jose Luis Navarro López-Montenegro (josnavlop4)



UNIVERSIDAD DE SEVILLA

EJERCICIO 1

```
1 package ejercicios.ejercicio1;
2
3 import java.util.ArrayList;
10
11 public record Ejercicio1Vertex(Integer index, List<Double> remaining)
12     implements VirtualVertex <Ejercicio1Vertex,Ejercicio1Edge,Integer> {
13
14     public static Ejercicio1Vertex of(Integer i, List<Double> rest) {
15         return new Ejercicio1Vertex(i, rest);
16     }
17
18     public static Ejercicio1Vertex initial() {
19         List<Double> lista = new ArrayList<>();
20         for(int j=0;j<DatosEjercicio1.getNumTipos();j++) {
21             lista.add(Double.valueOf(DatosEjercicio1.getKgdisp(j)));
22         }
23         return of(0,lista);
24     }
25
26     public static Predicate<Ejercicio1Vertex> goal(){
27         return v -> v.index() == DatosEjercicio1.getNumVariedades();
28     }
29
30     public static Predicate<Ejercicio1Vertex> goalHasSolution(){
31         return v -> v.index() == DatosEjercicio1.getNumVariedades();
32     }
33
34
35     @Override
36     public List<Integer> actions() {
37         List<Integer> alternativas = List2.emptyList();
38         if(index<DatosEjercicio1.getNumVariedades()) {
39             alternativas = List2.rangeList(0,
40                 DatosEjercicio1.getMinMax(index,remaining).intValue()+1);
41         }
42         return alternativas;
43     }
44
45     @Override
46     public Ejercicio1Vertex neighbor(Integer a) {
47         List<Double> res = new ArrayList<>(remaining);
48         for(int j = 0; j < remaining.size(); j++) {
49             Double kg = DatosEjercicio1.getKgxVariedad(index, j);
50             res.set(j, res.get(j) - a*kg);
51         }
52         return of(index + 1, res);
53     }
54
55     @Override
56     public Ejercicio1Edge edge(Integer a) {
57         return Ejercicio1Edge.of(this, neighbor(a), a);
58     }
59
60     public Ejercicio1Edge greedyEdge() {
61         return null;
62     }
63 }
```

```
1 package ejercicios.ejercicio1;
2
3 import _datos.DatosEjercicio1;
4
5
6 public record Ejercicio1Edge(Ejercicio1Vertex source, Ejercicio1Vertex target, Integer
7     action, Double weight)
8     implements SimpleEdgeAction <Ejercicio1Vertex, Integer> {
9
10    public static Ejercicio1Edge of(Ejercicio1Vertex s, Ejercicio1Vertex t, Integer a) {
11        double p = DatosEjercicio1.getBeneficio(s.index())*a;
12        return new Ejercicio1Edge(s, t, a, p);
13    }
14
15    @Override
16    public String toString() {
17        return String.format("%d; %.1f", action, weight);
18    }
19}
```

```
1 package ejercicios.ejercicio1;
2
3 import java.util.function.Predicate;
4
5
6 public class Ejercicio1Heuristic {
7
8     public static Double heuristic(Ejercicio1Vertex v1, Predicate<Ejercicio1Vertex> goal,
9         Ejercicio1Vertex v2) {
10         return IntStream.range(v1.index(), DatosEjercicio1.getNumVariedades())
11             .filter(i -> DatosEjercicio1.getMinMax(i,v1.remaining())>0)
12             .mapToDouble(i ->
13                 DatosEjercicio1.getBeneficio(i)*DatosEjercicio1.getMinMax(i,v1.remaining())).sum();
14     }
15 }
16
```

```
1 package tests;
2
3 import java.util.List;
10
11 public class TestEjercicio1 {
12
13     public static void main(String[] args) {
14
15         List.of(1,2,3).forEach(num_test -> {
16             TestsPI5.iniTest("Ejercicio1DatosEntrada", num_test, DatosEjercicio1::iniDatos);
17
18             // TODO Defina en el tipo vertice un m. factoria para el vertice inicial
19             // TODO Defina en el tipo vertice un m. static / Predicate para los vertices
20             finales
21                 TestsPI5.tests(
22                     Ejercicio1Vertex.initial(),           // Vertice Inicial
23                     Ejercicio1Vertex.goal(),            // Predicado para un vertice final
24                     GraphsPI5::multisetBuilder,          // Referencia al Builder del grafo
25                     Ejercicio1Vertex::greedyEdge,        // Referencia a la Funcion para la arista
26             voraz
27             solucion
28         });
29     }
30
31 }
32
```

```
1 package ejercicios.ejercicio1.manual;
2
3 import java.util.ArrayList;
9
10 public record Ejercicio1Problem(Integer index, List<Double> remaining) {
11
12     public static Ejercicio1Problem of(Integer i, List<Double> rest) {
13         return new Ejercicio1Problem(i, rest);
14     }
15
16     public static Ejercicio1Problem initial() {
17         List<Double> lista = new ArrayList<>();
18         for(int j=0;j<DatosEjercicio1.getNumTipos();j++) {
19             lista.add(DatosEjercicio1.getKgdisp(j).doubleValue());
20         }
21         return of(0,lista);
22     }
23
24     public List<Integer> actions() {
25         List<Integer> alternativas = new ArrayList<>();
26         if(index<DatosEjercicio1.getNumVariedades()) {
27             alternativas = List2.rangeList(0,
28                 DatosEjercicio1.getMinMax(index,remaining).intValue()+1);
29         }
30         return alternativas;
31     }
32
33     public Ejercicio1Problem neighbor(Integer a) {
34         List<Double> res = new ArrayList<>(remaining);
35         for(int j = 0; j < remaining.size(); j++) {
36             Double kg = DatosEjercicio1.getKgXVariedad(index, j);
37             res.set(j, res.get(j) - a*kg);
38         }
39         return of(index + 1, res);
40     }
41
42     public Double heuristic() {
43         return IntStream.range(index, DatosEjercicio1.getNumVariedades())
44             .filter(i -> DatosEjercicio1.getMinMax(i,remaining)>0)
45             .mapToDouble(i ->
46                 DatosEjercicio1.getBeneficio(i)*DatosEjercicio1.getMinMax(i,remaining)).sum();
47     }
47 }
```

```
1 package ejercicios.ejercicio1.manual;
2
3 import java.util.Comparator;
11
12 public class Ejercicio1PDR {
13
14     public static record Spm (Integer a, Integer weight) implements Comparable<Spm> {
15
16         public static Spm of(Integer a, Integer weight) {
17             return new Spm(a,weight);
18         }
19
20         @Override
21         public int compareTo(Spm o) {
22             return this.weight.compareTo(o.weight);
23         }
24     }
25
26     public static Map<Ejercicio1Problem, Spm> memory;
27     public static Integer mejorValor;
28
29     public static SolucionEjercicio1 search() {
30         memory = Map2.empty();
31         mejorValor=Integer.MIN_VALUE;
32
33         pdr_search(Ejercicio1Problem.initial(),0,memory);
34         return getSolucion();
35     }
36
37     private static Spm pdr_search(Ejercicio1Problem prob, Integer acumulado,
38         Map<Ejercicio1Problem, Spm> memory2) {
39         Spm res = null;
40         Boolean esTerminal = prob.index().equals(DatosEjercicio1.getNumVariedades());
41         Boolean esSolucion = prob.actions().isEmpty();
42
43         if(memory.containsKey(prob)) {
44             res = memory.get(prob);
45         } else if (esTerminal && esSolucion) {
46             res = Spm.of(null,0);
47             memory.put(prob, res);
48             if(acumulado > mejorValor) {
49                 mejorValor = acumulado;
50             }
51         } else {
52             List<Spm> soluciones = List2.empty();
53             for(Integer action : prob.actions()) {
54                 Double cota = acotar(acumulado,prob,action);
55                 if(cota < mejorValor) {
56                     continue;
57                 }
58                 Ejercicio1Problem vecino = prob.neighbor(action);
59                 Spm s= pdr_search(vecino, acumulado + action, memory);
60                 if(s!=null) {
61                     Spm amp = Spm.of(action, s.weight() + action);
62                     soluciones.add(amp);
63                 }
64                 res = soluciones.stream().max(Comparator.naturalOrder()).orElse(null);
65                 if(res!=null) memory.put(prob, res);
66             }
67         }
68     }
69 }
```

```
69
70
71     private static Double acotar(Integer acum, Ejercicio1Problem p, Integer a) {
72         return acum + a + p.neighbor(a).heuristic();
73     }
74
75     private static SolucionEjercicio1 getSolucion() {
76         List<Integer> acciones = List2.empty();
77         Ejercicio1Problem prob = Ejercicio1Problem.initial();
78         Spm spm = memory.get(prob);
79         while (spm!=null && spm.a!=null) {
80             Ejercicio1Problem old = prob;
81             acciones.add(spm.a);
82             prob=old.neighbor(spm.a);
83             spm = memory.get(prob);
84         }
85         return SolucionEjercicio1.of(acciones);
86     }
87 }
88 }
```

```
1 package tests.manual;
2
3 import java.util.List;
9
10 public class TestEjercicio1PDR {
11
12     public static void main(String[] args) {
13
14         List.of(1,2,3).forEach(num_test -> {
15             DatosEjercicio1.iniDatos("ficheros/Ejercicio1DatosEntrada"+num_test+".txt");
16             String2.toConsole("Solucion obtenida: %s\n", Ejercicio1PDR.search());
17             TestsPI5.line("*");
18     });
19 }
20
21 }
22
```

RESULTADOS GRAFOS VIRTUALES

```
Tipos: [Tipo[nombre=C01, kgdisp=5], Tipo[nombre=C02, kgdisp=4], Tipo[nombre=C03, kgdisp=1], Tipo[nombre=C04, kgdisp=2], Tipo[nombre=C05, kgdisp=8],  
Tipo[nombre=C06, kgdisp=1]]  
Variedades = {  
    Variedad[nombre=P01, beneficio=20, comp=[0.5, 0.4, 0.1, 0.0, 0.0, 0.0]]  
    Variedad[nombre=P02, beneficio=10, comp=[0.0, 0.0, 0.0, 0.2, 0.8, 0.0]]  
    Variedad[nombre=P03, beneficio=5, comp=[0.0, 0.0, 0.0, 0.0, 0.0, 1.0]]  
}  
=====  
Voraz no obtuvo solucion  
[10, 10, 1]  
Solucion A*: Kg que deben producirse para maximizar beneficios:  
P01: 10 kgs  
P03: 1 kgs  
P02: 10 kgs  
Beneficio: 305.0  
Path de la solucion: [10, 10, 1]  
[10, 10, 1]  
Solucion PDR: Kg que deben producirse para maximizar beneficios:  
P01: 10 kgs  
P03: 1 kgs  
P02: 10 kgs  
Beneficio: 305.0  
Path de la solucion: [10, 10, 1]  
[10, 10, 1]  
Solucion BT: Kg que deben producirse para maximizar beneficios:  
P01: 10 kgs  
P03: 1 kgs  
P02: 10 kgs  
Beneficio: 305.0  
Path de la solucion: [10, 10, 1]  
*****  
Tipos: [Tipo[nombre=C01, kgdisp=11], Tipo[nombre=C02, kgdisp=9], Tipo[nombre=C03, kgdisp=7], Tipo[nombre=C04, kgdisp=12], Tipo[nombre=C05, kgdisp=6]]  
Variedades = {  
    Variedad[nombre=P01, beneficio=20, comp=[0.2, 0.4, 0.0, 0.0, 0.0, 0.0]]  
    Variedad[nombre=P02, beneficio=10, comp=[0.0, 0.3, 0.7, 0.0, 0.0]]  
    Variedad[nombre=P03, beneficio=80, comp=[0.4, 0.0, 0.0, 0.6, 0.0]]  
}  
=====  
Voraz no obtuvo solucion  
[15, 10, 20]  
Solucion A*: Kg que deben producirse para maximizar beneficios:  
P01: 15 kgs  
P02: 10 kgs  
P03: 20 kgs  
Beneficio: 2000.0  
Path de la solucion: [15, 10, 20]  
[15, 10, 20]  
Solucion PDR: Kg que deben producirse para maximizar beneficios:  
P01: 15 kgs  
P02: 10 kgs  
P03: 20 kgs  
Beneficio: 2000.0  
Path de la solucion: [15, 10, 20]  
[15, 10, 20]  
Solucion BT: Kg que deben producirse para maximizar beneficios:  
P01: 15 kgs  
P02: 10 kgs  
P03: 20 kgs  
Beneficio: 2000.0  
Path de la solucion: [15, 10, 20]  
*****  
Tipos: [Tipo[nombre=C01, kgdisp=35], Tipo[nombre=C02, kgdisp=4], Tipo[nombre=C03, kgdisp=12], Tipo[nombre=C04, kgdisp=5], Tipo[nombre=C05, kgdisp=30],  
Tipo[nombre=C06, kgdisp=42], Tipo[nombre=C07, kgdisp=3], Tipo[nombre=C08, kgdisp=2], Tipo[nombre=C09, kgdisp=20], Tipo[nombre=C10, kgdisp=3]]  
Variedades = {  
    Variedad[nombre=P01, beneficio=60, comp=[0.5, 0.0, 0.4, 0.0, 0.0, 0.0, 0.1, 0.0, 0.0, 0.0]]  
    Variedad[nombre=P02, beneficio=25, comp=[0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]]  
    Variedad[nombre=P03, beneficio=5, comp=[0.0, 0.4, 0.0, 0.0, 0.0, 0.0, 0.8, 0.0, 0.0, 0.0]]  
    Variedad[nombre=P04, beneficio=25, comp=[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.8, 0.0, 0.0, 0.2]]  
    Variedad[nombre=P05, beneficio=15, comp=[0.0, 0.0, 0.4, 0.0, 0.0, 0.0, 0.0, 0.6, 0.0, 0.0]]  
    Variedad[nombre=P06, beneficio=100, comp=[0.2, 0.0, 0.0, 0.0, 0.3, 0.0, 0.0, 0.2, 0.0]]  
}  
=====  
Voraz no obtuvo solucion  
[30, 4, 0, 15, 0, 100]  
Solucion A*: Kg que deben producirse para maximizar beneficios:  
P06: 100 kgs  
P01: 30 kgs  
P02: 4 kgs  
P04: 15 kgs  
Beneficio: 12275.0  
Path de la solucion: [30, 4, 0, 15, 0, 100]  
[30, 4, 0, 15, 0, 100]  
Solucion PDR: Kg que deben producirse para maximizar beneficios:  
P06: 100 kgs  
P01: 30 kgs  
P02: 4 kgs  
P04: 15 kgs  
Beneficio: 12275.0  
Path de la solucion: [30, 4, 0, 15, 0, 100]  
[30, 4, 0, 15, 0, 100]  
Solucion BT: Kg que deben producirse para maximizar beneficios:  
P06: 100 kgs  
P01: 30 kgs  
P02: 4 kgs  
P04: 15 kgs  
Beneficio: 12275.0  
Path de la solucion: [30, 4, 0, 15, 0, 100]  
*****
```

RESULTADOS MANUAL

```
Tipos: [Tipo[nombre=C01, kgdisp=5], Tipo[nombre=C02, kgdisp=4], Tipo[nombre=C03, kgdisp=1], Tipo[nombre=C04, kgdisp=2],  
Tipo[nombre=C05, kgdisp=8], Tipo[nombre=C06, kgdisp=1]]  
Variedades = {  
    Variedad[nombre=P01, beneficio=20, comp=[0.5, 0.4, 0.1, 0.0, 0.0, 0.0]]  
    Variedad[nombre=P02, beneficio=10, comp=[0.0, 0.0, 0.0, 0.2, 0.8, 0.0]]  
    Variedad[nombre=P03, beneficio=5, comp=[0.0, 0.0, 0.0, 0.0, 0.0, 1.0]]  
}
```

Solucion obtenida: Kg que deben producirse para maximizar beneficios:

P01: 10 kgs

P03: 1 kgs

P02: 10 kgs

Beneficio: 305.0

```
*****  
Tipos: [Tipo[nombre=C01, kgdisp=11], Tipo[nombre=C02, kgdisp=9], Tipo[nombre=C03, kgdisp=7], Tipo[nombre=C04, kgdisp=12],  
Tipo[nombre=C05, kgdisp=6]]  
Variedades = {  
    Variedad[nombre=P01, beneficio=20, comp=[0.2, 0.4, 0.0, 0.0, 0.0, 0.4]]  
    Variedad[nombre=P02, beneficio=10, comp=[0.0, 0.3, 0.7, 0.0, 0.0, 0.0]]  
    Variedad[nombre=P03, beneficio=80, comp=[0.4, 0.0, 0.0, 0.6, 0.0, 0.0]]  
}
```

Solucion obtenida: Kg que deben producirse para maximizar beneficios:

P01: 15 kgs

P02: 10 kgs

P03: 20 kgs

Beneficio: 2000.0

```
*****  
Tipos: [Tipo[nombre=C01, kgdisp=35], Tipo[nombre=C02, kgdisp=4], Tipo[nombre=C03, kgdisp=12], Tipo[nombre=C04, kgdisp=5],  
Tipo[nombre=C05, kgdisp=30], Tipo[nombre=C06, kgdisp=42], Tipo[nombre=C07, kgdisp=3], Tipo[nombre=C08, kgdisp=2],  
Tipo[nombre=C09, kgdisp=20], Tipo[nombre=C10, kgdisp=3]]  
Variedades = {  
    Variedad[nombre=P01, beneficio=60, comp=[0.5, 0.0, 0.4, 0.0, 0.0, 0.0, 0.1, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]]  
    Variedad[nombre=P02, beneficio=25, comp=[0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]]  
    Variedad[nombre=P03, beneficio=5, comp=[0.0, 0.4, 0.0, 0.0, 0.0, 0.0, 0.0, 0.8, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]]  
    Variedad[nombre=P04, beneficio=25, comp=[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.8, 0.0, 0.0, 0.0, 0.0, 0.2]]  
    Variedad[nombre=P05, beneficio=15, comp=[0.0, 0.0, 0.4, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.6, 0.0, 0.0, 0.0, 0.0]]  
    Variedad[nombre=P06, beneficio=100, comp=[0.2, 0.0, 0.0, 0.0, 0.3, 0.0, 0.0, 0.0, 0.0, 0.2, 0.0, 0.0, 0.0, 0.0]]  
}
```

Solucion obtenida: Kg que deben producirse para maximizar beneficios:

P06: 100 kgs

P01: 30 kgs

P02: 4 kgs

P04: 15 kgs

Beneficio: 12275.0

EJERCICIO 2

```
1 package ejercicios.ejercicio2;
2
3 import java.util.ArrayList;
12
13 public record Ejercicio2Vertex(Integer index, Set<Integer> remaining, Set<Integer> centers)
14     implements VirtualVertex<Ejercicio2Vertex, Ejercicio2Edge, Integer>{
15
16     public static Ejercicio2Vertex of(Integer i, Set<Integer> rest, Set<Integer> ce) {
17         return new Ejercicio2Vertex(i, rest,ce);
18     }
19
20     public static Ejercicio2Vertex initial() {
21         return of(0,Set2.copy(DatosEjercicio2.getTematica()),Set2.empty());
22     }
23
24     public static Predicate<Ejercicio2Vertex> goal(){
25         return v->v.index() == DatosEjercicio2.getN();
26     }
27
28     public static Predicate<Ejercicio2Vertex> goalHasSolution(){
29         return v->v.remaining.isEmpty();
30     }
31
32     @Override
33     public List<Integer> actions() {
34         List<Integer> alternativas = new ArrayList<>();
35
36         if (index < DatosEjercicio2.getN()) {
37             if(remaining.isEmpty()) {
38                 alternativas.add(0);
39             } else {
40                 if(centers.size()<DatosEjercicio2.getMaxC() || centers.contains
41                     (DatosEjercicio2.getCentroXCurso(index))){
42                     alternativas = List.of(0,1);
43                 }else {
44                     alternativas.add(0);
45                 }
46             }
47         }
48         return alternativas;
49     }
50
51     @Override
52     public Ejercicio2Vertex neighbor(Integer a) {
53         Set<Integer> rest = a==0? Set2.copy(remaining):Set2.difference(remaining,
54             DatosEjercicio2.getTematicaXCurso(index));
55         Set<Integer> cent = new HashSet<>(centers);
56         if(a==1) {
57             cent.add(DatosEjercicio2.getCentroXCurso(index));
58         }
59         return of(index+1,rest,cent);
60     }
61
62     @Override
63     public Ejercicio2Edge edge(Integer a) {
64         return Ejercicio2Edge.of(this, neighbor(a), a);
65     }
66
67     public Ejercicio2Edge greedyEdge() {
68         return null;
69     }
70 }
```

```
1 package ejercicios.ejercicio2;
2
3 import _datos.DatosEjercicio2;
4
5
6 public record Ejercicio2Edge(Ejercicio2Vertex source, Ejercicio2Vertex target, Integer
7     action, Double weight)
8     implements SimpleEdgeAction<Ejercicio2Vertex, Integer>{
9
10    public static Ejercicio2Edge of(Ejercicio2Vertex s, Ejercicio2Vertex t, Integer a) {
11        double p = DatosEjercicio2.getCoste(s.index());
12        return new Ejercicio2Edge (s,t,a,a*p);
13    }
14
15    @Override
16    public String toString() {
17        return String.format("%d; %.1f", action, weight);
18    }
19}
20
```

```
1 package ejercicios.ejercicio2;
2
3 import java.util.function.Predicate;
7
8 public class Ejercicio2Heuristic {
9
10    public static Double heuristic(Ejercicio2Vertex v1, Predicate<Ejercicio2Vertex> goal,
11                                    Ejercicio2Vertex v2) {
12        return v1.remaining().isEmpty() ? 0.:
13            IntStream.range(v1.index(), DatosEjercicio2.getN())
14                .filter( i-> v1.centers().size() < DatosEjercicio2.getMaxC())
15                .mapToDouble(i -> DatosEjercicio2.getCoste(i)).min().orElse(100.);
16    }
17}
```

```
1 package tests;
2
3 import java.util.List;
10
11 public class TestEjercicio2 {
12
13     public static void main(String[] args) {
14         List.of(1,2,3).forEach(num_test -> {
15             TestsPI5.iniTest("Ejercicio2DatosEntrada", num_test, DatosEjercicio2::iniDatos);
16
17             // TODO Defina en el tipo vertice un m. factoria para el vertice inicial
18             // TODO Defina en el tipo vertice un m. static / Predicate para los vertices
19             finales
20             TestsPI5.tests(
21                 Ejercicio2Vertex.initial(),           // Vertice Inicial
22                 Ejercicio2Vertex.goal(),            // Predicado para un vertice final
23                 GraphsPI5::multisetBuilder,          // Referencia al Builder del grafo
24                 Ejercicio2Vertex::greedyEdge,        // Referencia a la Funcion para la arista
25                 voraz
26                 solucion
27             });
28
29 }
30
```

```
1 package ejercicios.ejercicio2.manual;
2
3 import java.util.ArrayList;
11
12 public record Ejercicio2Problem(Integer index, Set<Integer> remaining, Set<Integer> centers) {
13
14     public static Ejercicio2Problem of(Integer i, Set<Integer> rest, Set<Integer> ce) {
15         return new Ejercicio2Problem(i, rest, ce);
16     }
17
18     public static Ejercicio2Problem initial() {
19         return of(0, Set2.copy(DatosEjercicio2.getTematica()), Set2.empty());
20     }
21
22     public List<Integer> actions() {
23         List<Integer> alternativas = new ArrayList<>();
24
25         if (index < DatosEjercicio2.getN()) {
26             if (remaining.isEmpty()) {
27                 alternativas.add(0);
28             } else {
29                 if (centers.size() < DatosEjercicio2.getMaxC() || centers.contains
(DatosEjercicio2.getCentroXCurso(index))) {
30                     alternativas = List.of(0, 1);
31                 } else {
32                     alternativas.add(0);
33                 }
34             }
35         }
36         return alternativas;
37     }
38
39     public Ejercicio2Problem neighbor(Integer a) {
40         Set<Integer> rest = a==0? Set2.copy(remaining):Set2.difference(remaining,
DatosEjercicio2.getTematicaXCurso(index));
41         Set<Integer> cent = new HashSet<>(centers);
42         if(a==1) {
43             cent.add(DatosEjercicio2.getCentroXCurso(index));
44         }
45         return of(index+1, rest, cent);
46     }
47
48     public Double heuristic() {
49         return remaining.isEmpty()? 0.:
50             IntStream.range(index, DatosEjercicio2.getN())
51                 .filter( i-> centers.size() < DatosEjercicio2.getMaxC())
52                 .mapToDouble(i -> DatosEjercicio2.getCoste(i)).min().orElse(100.);
53     }
54
55 }
```

```
1 package ejercicios.ejercicio2.manual;
2
3 import java.util.Comparator;
11
12 public class Ejercicio2PDR {
13
14     public static record Spm (Integer a, Integer weight) implements Comparable<Spm> {
15
16         public static Spm of(Integer a, Integer weight) {
17             return new Spm(a,weight);
18         }
19
20         @Override
21         public int compareTo(Spm o) {
22             return this.weight.compareTo(o.weight);
23         }
24     }
25
26     public static Map<Ejercicio2Problem, Spm> memory;
27     public static Integer mejorValor;
28
29     public static SolucionEjercicio2 search() {
30         memory = Map2.empty();
31         mejorValor=Integer.MAX_VALUE;
32
33         pdr_search(Ejercicio2Problem.initial(),0,memory);
34         return getSolucion();
35     }
36
37     private static Spm pdr_search(Ejercicio2Problem prob, Integer acumulado,
38         Map<Ejercicio2Problem, Spm> memory2) {
39         Spm res = null;
40         Boolean esTerminal = prob.index().equals(DatosEjercicio2.getN());
41         Boolean esSolucion = prob.remaining().isEmpty();
42
43         if(memory.containsKey(prob)) {
44             res = memory.get(prob);
45         } else if (esTerminal && esSolucion) {
46             res = Spm.of(null,0);
47             memory.put(prob, res);
48             if(acumulado < mejorValor) {
49                 mejorValor = acumulado;
50             }
51         } else {
52             List<Spm> soluciones = List2.empty();
53             for(Integer action : prob.actions()) {
54                 Double cota = acotar(acumulado,prob,action);
55                 if(cota > mejorValor) {
56                     continue;
57                 }
58                 Ejercicio2Problem vecino = prob.neighbor(action);
59                 Spm s= pdr_search(vecino, acumulado + action, memory);
60                 if(s!=null) {
61                     Spm amp = Spm.of(action, s.weight() + action);
62                     soluciones.add(amp);
63                 }
64                 res = soluciones.stream().min(Comparator.naturalOrder()).orElse(null);
65                 if(res!=null) memory.put(prob, res);
66             }
67         }
68     }
69 }
```

```
69
70
71     private static Double acotar(Integer acum, Ejercicio2Problem p, Integer a) {
72         return acum + a + p.neighbor(a).heuristic();
73     }
74
75     private static SolucionEjercicio2 getSolucion() {
76         List<Integer> acciones = List2.empty();
77         Ejercicio2Problem prob = Ejercicio2Problem.initial();
78         Spm spm = memory.get(prob);
79         while (spm!=null && spm.a!=null) {
80             Ejercicio2Problem old = prob;
81             acciones.add(spm.a);
82             prob=old.neighbor(spm.a);
83             spm = memory.get(prob);
84         }
85         return SolucionEjercicio2.of(acciones);
86     }
87
88 }
89
```

```
1 package tests.manual;
2
3 import java.util.List;
9
10 public class TestEjercicio2PDR {
11
12     public static void main(String[] args) {
13         List.of(1,2,3).forEach(num_test -> {
14             DatosEjercicio2.iniDatos("ficheros/Ejercicio2DatosEntrada"+num_test+".txt");
15             String2.toConsole("Solucion obtenida: %s\n", Ejercicio2PDR.search());
16             TestsPI5.line("*");
17         });
18     }
19
20 }
21
```

RESULTADOS GRAFOS VIRTUALES

```
cursos: [Curso[id=0, tematicas=[1, 2, 3, 4], coste=10.0, centro=0], Curso[id=1, tematicas=[1, 4], coste=3.0, centro=0], Curso[id=2, tematicas=[5], coste=1.5, centro=1], Curso[id=3, tematicas=[5], coste=5.0, centro=0]]  
Max_Centros:1  
=====  
Voraz no obtuvo solucion  


---

[1, 0, 0, 1]  
Solucion A*: Cursos elegidos: {S0, S3}  
Coste Total: 15,0  
Path de la solucion: [1, 0, 0, 1]  


---

[1, 0, 0, 1]  
Solucion PDR: Cursos elegidos: {S0, S3}  
Coste Total: 15,0  
Path de la solucion: [1, 0, 0, 1]  


---

[1, 0, 0, 1]  
Solucion BT: Cursos elegidos: {S0, S3}  
Coste Total: 15,0  
Path de la solucion: [1, 0, 0, 1]  
*****  
cursos: [Curso[id=0, tematicas=[2, 3], coste=2.0, centro=0], Curso[id=1, tematicas=[4], coste=3.0, centro=0], Curso[id=2, tematicas=[1, 5], coste=5.0, centro=0], Curso[id=3, tematicas=[1, 3, 4], coste=3.5, centro=2], Curso[id=4, tematicas=[4, 5], coste=1.5, centro=1]]  
Max_Centros:2  
=====  
Voraz no obtuvo solucion  


---

[1, 0, 1, 0, 1]  
Solucion A*: Cursos elegidos: {S0, S2, S4}  
Coste Total: 8,5  
Path de la solucion: [1, 0, 1, 0, 1]  


---

[1, 0, 1, 0, 1]  
Solucion PDR: Cursos elegidos: {S0, S2, S4}  
Coste Total: 8,5  
Path de la solucion: [1, 0, 1, 0, 1]  


---

[1, 0, 1, 0, 1]  
Solucion BT: Cursos elegidos: {S0, S2, S4}  
Coste Total: 8,5  
Path de la solucion: [1, 0, 1, 0, 1]  
*****  
cursos: [Curso[id=0, tematicas=[2, 6, 7], coste=2.0, centro=2], Curso[id=1, tematicas=[7], coste=3.0, centro=0], Curso[id=2, tematicas=[1, 5], coste=5.0, centro=0], Curso[id=3, tematicas=[1, 3, 4], coste=3.5, centro=2], Curso[id=4, tematicas=[3, 7], coste=1.5, centro=1], Curso[id=5, tematicas=[4, 5, 6], coste=4.5, centro=0], Curso[id=6, tematicas=[5, 6], coste=6.0, centro=1], Curso[id=7, tematicas=[2, 3, 5], coste=1.0, centro=1]]  
Max_Centros:3  
=====  
Voraz no obtuvo solucion  


---

[1, 0, 0, 1, 0, 0, 1]  
Solucion A*: Cursos elegidos: {S0, S3, S7}  
Coste Total: 6,5  
Path de la solucion: [1, 0, 0, 1, 0, 0, 0, 1]  


---

[1, 0, 0, 1, 0, 0, 1]  
Solucion PDR: Cursos elegidos: {S0, S3, S7}  
Coste Total: 6,5  
Path de la solucion: [1, 0, 0, 1, 0, 0, 0, 1]  


---

[1, 0, 0, 1, 0, 0, 1]  
Solucion BT: Cursos elegidos: {S0, S3, S7}  
Coste Total: 6,5  
Path de la solucion: [1, 0, 0, 1, 0, 0, 0, 1]  
*****
```

RESULTADOS MANUAL

```
cursos: [Curso[id=0, tematicas=[1, 2, 3, 4], coste=10.0, centro=0], Curso[id=1, tematicas=[1, 4], coste=3.0, centro=0], Curso[id=2, tematicas=[5], coste=1.5, centro=1], Curso[id=3, tematicas=[5], coste=5.0, centro=0]]  
Max_Centros:1  
[1, 0, 0, 1]  
Solucion obtenida: Cursos elegidos: {S0, S3}  
Coste Total: 15,0  
*****  
cursos: [Curso[id=0, tematicas=[2, 3], coste=2.0, centro=0], Curso[id=1, tematicas=[4], coste=3.0, centro=0], Curso[id=2, tematicas=[1, 5], coste=5.0, centro=0], Curso[id=3, tematicas=[1, 3, 4], coste=3.5, centro=2], Curso[id=4, tematicas=[4, 5], coste=1.5, centro=1]]  
Max_Centros:2  
[1, 0, 1, 0, 1]  
Solucion obtenida: Cursos elegidos: {S0, S2, S4}  
Coste Total: 8,5  
*****  
cursos: [Curso[id=0, tematicas=[2, 6, 7], coste=2.0, centro=2], Curso[id=1, tematicas=[7], coste=3.0, centro=0], Curso[id=2, tematicas=[1, 5], coste=5.0, centro=0], Curso[id=3, tematicas=[1, 3, 4], coste=3.5, centro=2], Curso[id=4, tematicas=[3, 7], coste=1.5, centro=1], Curso[id=5, tematicas=[4, 5, 6], coste=4.5, centro=0], Curso[id=6, tematicas=[5, 6], coste=6.0, centro=1], Curso[id=7, tematicas=[2, 3, 5], coste=1.0, centro=1]]  
Max_Centros:3  
[1, 0, 0, 1, 0, 0, 1]  
Solucion obtenida: Cursos elegidos: {S0, S3, S7}  
Coste Total: 6,5  
*****
```

EJERCICIO 3

```
1 package ejercicios.ejercicio3;
2
3 import java.util.ArrayList;
9
10 public record Ejercicio3Vertex(Integer index, List<Integer> days, List<List<Integer>>
11     distribucion)
12     implements VirtualVertex<Ejercicio3Vertex, Ejercicio3Edge, Integer>{
13
14     public static Ejercicio3Vertex of(Integer z, List<Integer> d, List<List<Integer>> dist)
15     {
16         return new Ejercicio3Vertex(z, d, dist);
17     }
18
19     public static Ejercicio3Vertex initial() {
20
21         List<Integer> d = new ArrayList<>();
22         List<List<Integer>> dist = new ArrayList<>();
23
24         for(int i=0;i<DatosEjercicio3.getN();i++) {
25             d.add(DatosEjercicio3.getDiasDispl(i));
26         }
27         for(int j=0;j<DatosEjercicio3.getM();j++) {
28             List<Integer> especialidades = new ArrayList<>();
29             for(int k=0;k<DatosEjercicio3.getE();k++) {
30                 especialidades.add(DatosEjercicio3.getDiasNecesarios(j, k));
31             }
32             dist.add(especialidades);
33         }
34     }
35
36     public static Predicate<Ejercicio3Vertex> goal(){
37         return v->v.index() == DatosEjercicio3.getM()*DatosEjercicio3.getN();
38     }
39
40     public static Predicate<Ejercicio3Vertex> goalHasSolution(){
41         return v->v.actions().size()==0;
42     }
43
44     @Override
45     public List<Integer> actions() {
46         List<Integer> alternativas = new ArrayList<>();
47
48         Integer m = DatosEjercicio3.getM();
49         Integer n = DatosEjercicio3.getN();
50
51         if(index<(m*n)) {
52             Integer i = index/m;
53             Integer j = index%m;
54             Integer k = DatosEjercicio3.getEspInv(i);
55
56             for(int a=0;a<=days.get(i) && a<=distribucion.get(j).get(k);a++) {
57                 alternativas.add(a);
58             }
59         }
60     }
61     return alternativas;
62 }
63
64     @Override
65     public Ejercicio3Vertex neighbor(Integer a) {
```

```
66     Integer m = DatosEjercicio3.getM();
67     Integer i = index/m;
68     Integer j = index%m;
69     Integer k = DatosEjercicio3.getEspInv(i);
70
71     List<List<Integer>> dist = new ArrayList<>();
72     for(List<Integer> dis : distribucion) {
73         dist.add(new ArrayList<>(dis));
74     }
75
76     List<Integer> d = new ArrayList<>(days);
77
78     d.set(i, days.get(i)-a);
79     dist.get(j).set(k, dist.get(j).get(k)-a);
80
81     return of(index+1,d,dist);
82 }
83
84
85 @Override
86 public Ejercicio3Edge edge(Integer a) {
87     return Ejercicio3Edge.of(this, neighbor(a), a);
88 }
89
90 public Ejercicio3Edge greedyEdge() {
91     return null;
92 }
93 }
94 }
```

```
1 package ejercicios.ejercicio3;
2
3 import _datos.DatosEjercicio3;
4
5
6 public record Ejercicio3Edge(Ejercicio3Vertex source, Ejercicio3Vertex target, Integer
7     action, Double weight)
8 implements SimpleEdgeAction<Ejercicio3Vertex, Integer> {
9
10    public static Ejercicio3Edge of(Ejercicio3Vertex s, Ejercicio3Vertex t, Integer a) {
11        Integer j = s.index()%DatosEjercicio3.getM();
12        Integer i = s.index()/DatosEjercicio3.getM();
13
14        double p = 0.;
15        Boolean realizado = true;
16        if(i==(DatosEjercicio3.getN()-1)) {
17
18            for(int k=0; k<DatosEjercicio3.getE();k++) {
19                if(t.distribucion().get(j).get(k)!=0) {
20                    realizado=false;
21                    break;
22                }
23            }
24            if (realizado) {
25                p = DatosEjercicio3.getCalidadTrbj(j);
26            }
27        }
28
29        return new Ejercicio3Edge (s,t,a,p);
30    }
31
32    @Override
33    public String toString() {
34        return String.format("%d; %.1f", action, weight);
35    }
36
37 }
38
```

```
1 package ejercicios.ejercicio3;
2
3 import java.util.function.Predicate;
4
5
6 public class Ejercicio3Heuristic {
7
8     public static Double heuristic(Ejercicio3Vertex v1, Predicate<Ejercicio3Vertex> goal,
9         Ejercicio3Vertex v2) {
10         return v1.days().isEmpty() ? 0.:IntStream.range(v1.index(),
11             DatosEjercicio3.getN()*DatosEjercicio3.getM())
12             .mapToDouble(z ->
13                 DatosEjercicio3.getCalidadTrbj(z%DatosEjercicio3.getM())).sum();
14     }
15 }
16
```

```
1 package tests;
2
3 import java.util.List;
10
11 public class TestEjercicio3 {
12
13     public static void main(String[] args) {
14
15         List.of(1,2,3).forEach(num_test -> {
16             TestsPI5.iniTest("Ejercicio3DatosEntrada", num_test, DatosEjercicio3::iniDatos);
17
18             // TODO Defina en el tipo vertice un m. factoria para el vertice inicial
19             // TODO Defina en el tipo vertice un m. static / Predicate para los vertices
20             finales
21                 TestsPI5.tests(Ejercicio3Vertex.initial(),           // Vertice Inicial
22                               Ejercicio3Vertex.goal(),            // Predicado para un vertice final
23                               GraphsPI5::multisetBuilder,        // Referencia al Builder del grafo
24                               Ejercicio3Vertex::greedyEdge,      // Referencia a la Funcion para la arista
25             voraz
26                 SolucionEjercicio3::of);          // Referencia al metodo factoria para la
27             solucion
28         });
29     }
30 }
```

```
1 package ejercicios.ejercicio3.manual;
2
3 import java.util.ArrayList;
4
5 public record Ejercicio3Problem(Integer index, List<Integer> days, List<List<Integer>>
6     distribucion) {
7
8     public static Ejercicio3Problem of(Integer z, List<Integer> d, List<List<Integer>> dist)
9     {
10         return new Ejercicio3Problem(z, d, dist);
11     }
12 }
13
14
15     public static Ejercicio3Problem initial() {
16
17         List<Integer> d = new ArrayList<>();
18         List<List<Integer>> dist = new ArrayList<>();
19
20         for(int i=0;i<DatosEjercicio3.getN();i++) {
21             d.add(DatosEjercicio3.getDiasDispl(i));
22         }
23         for(int j=0;j<DatosEjercicio3.getM();j++) {
24             List<Integer> especialidades = new ArrayList<>();
25             for(int k=0;k<DatosEjercicio3.getE();k++) {
26                 especialidades.add(DatosEjercicio3.getDiasNecesarios(j, k));
27             }
28             dist.add(especialidades);
29         }
30
31         return of(0,d,dist);
32     }
33
34     public List<Integer> actions() {
35         List<Integer> alternativas = new ArrayList<>();
36
37         Integer m = DatosEjercicio3.getM();
38         Integer n = DatosEjercicio3.getN();
39
40
41         if(index<(m*n)) {
42             Integer i = index/m;
43             Integer j = index%m;
44             Integer k = DatosEjercicio3.getEspInv(i);
45
46             for(int a=0;a<=days.get(i) && a<=distribucion.get(j).get(k);a++) {
47                 alternativas.add(a);
48             }
49         }
50         return alternativas;
51     }
52
53     public Ejercicio3Problem neighbor(Integer a) {
54
55         Integer m = DatosEjercicio3.getM();
56         Integer i = index/m;
57         Integer j = index%m;
58         Integer k = DatosEjercicio3.getEspInv(i);
59
60         List<List<Integer>> dist = new ArrayList<>();
61         for(List<Integer> dis : distribucion) {
62             dist.add(new ArrayList<>(dis));
63         }
64     }
```

```
65     List<Integer> d = new ArrayList<>(days);
66
67     d.set(i, days.get(i)-a);
68     dist.get(j).set(k, dist.get(j).get(k)-a);
69
70     return of(index+1,d,dist);
71 }
72
73 public Double heuristic() {
74     return IntStream.range(index, DatosEjercicio3.getN()*DatosEjercicio3.getM())
75 //             .filter(z->DatosEjercicio3.getEpsdelTrbj(z%DatosEjercicio3.getM()).contains
76 (DatosEjercicio3.getEspInv(z/DatosEjercicio3.getM())))
77             .mapToDouble(z ->
78     DatosEjercicio3.getCalidadTrbj(z%DatosEjercicio3.getM())).max().orElse(-10.);
79 }
80
```

```
1 package ejercicios.ejercicio3.manual;
2
3 import java.util.ArrayList;
4
5 public class Ejercicio3State {
6
7     Ejercicio3Problem actual;
8     Double acumulado;
9     List<Integer> acciones;
10    List<Ejercicio3Problem> anteriores;
11
12    private Ejercicio3State(Ejercicio3Problem p, Double a, List<Integer> ls1,
13     List<Ejercicio3Problem> ls2) {
14        actual = p;
15        acumulado = a;
16        acciones = ls1;
17        anteriores = ls2;
18    }
19
20    public static Ejercicio3State initial() {
21        Ejercicio3Problem pi = Ejercicio3Problem.initial();
22        return of(pi,0.,new ArrayList<>(),new ArrayList<>());
23    }
24
25    public static Ejercicio3State of(Ejercicio3Problem prob, Double acum, List<Integer> lsa,
26     List<Ejercicio3Problem> lsp) {
27        return new Ejercicio3State(prob, acum, lsa, lsp);
28    }
29
30    public static Integer trabajoCompletado(Ejercicio3Problem s, Ejercicio3Problem t) {
31        Integer conteoS = 0;
32        Integer conteoT = 0;
33        Integer j = s.index()%DatosEjercicio3.getM();
34
35        conteoT = t.distribucion().get(j).stream().mapToInt(i->i).sum();
36        conteoS = s.distribucion().get(j).stream().mapToInt(i->i).sum();
37
38        return conteoT==0 && conteoS!=0?1:0;
39    }
40
41    public void forward(Integer a) {
42        acumulado += DatosEjercicio3.getCalidadTrbj(actual.index()%DatosEjercicio3.getM());
43        trabajoCompletado(actual, actual.neighbor(a));
44        acciones.add(a);
45        anteriores.add(actual);
46        actual = actual.neighbor(a);
47    }
48
49    public void back() {
50        int last = acciones.size()-1;
51        var prob_ant = anteriores.get(last);
52
53        acumulado -= DatosEjercicio3.getCalidadTrbj(prob_ant.index()%DatosEjercicio3.getM());
54        * trabajoCompletado(prob_ant, actual) ;
55        acciones.remove(last);
56        anteriores.remove(last);
57        actual = prob_ant;
58    }
59
60    public List<Integer> alternativas() {
61        return actual.actions();
62    }
63}
```

```
63
64     public Double cota(Integer a) {
65         Integer j = actual.index()%DatosEjercicio3.getM();
66
67         return acumulado + DatosEjercicio3.getCalidadTrbj(j) * trabajoCompletado(actual,
68             actual.neighbor(a)) + actual.neighbor(a).heuristic();
69     }
70
71     public Boolean esSolucion() {
72         return actual.index() == DatosEjercicio3.getM()*DatosEjercicio3.getN();
73     }
74
75     public Boolean esTerminal() {
76         return actual.index() == DatosEjercicio3.getM()*DatosEjercicio3.getN();
77     }
78
79     public SolucionEjercicio3 getSolucion() {
80         return SolucionEjercicio3.of(acciones);
81     }
82 }
```

```
1 package ejercicios.ejercicio3.manual;
2
3 import java.util.HashSet;
7
8 public class Ejercicio3BT {
9
10    private static Double mejorValor;
11    private static Ejercicio3State estado;
12    private static Set<SolucionEjercicio3> soluciones;
13
14    public static void search() {
15        soluciones = new HashSet<>();
16        mejorValor = Double.MIN_VALUE;
17        estado = Ejercicio3State.initial();
18        bt_search();
19    }
20
21    private static void bt_search() {
22        if (estado.esSolucion()) {
23            Double valorObtenido = estado.acumulado;
24            if (valorObtenido > mejorValor) {
25                mejorValor=valorObtenido;
26                soluciones.add(estado.getSolucion());
27            }
28        } else if (!estado.esTerminal()) {
29            for (Integer a: estado.alternativas()) {
30                if (estado.cota(a) >= mejorValor) {
31                    estado.forward(a);
32                    bt_search();
33                    estado.back();
34                }
35            }
36        }
37    }
38
39    public static Set<SolucionEjercicio3> getSoluciones() {
40        return soluciones;
41    }
42
43 }
44
```

```
1 package tests.manual;
2
3 import java.util.List;
9
10 public class TestEjercicio3BT {
11
12     public static void main(String[] args) {
13         List.of(1,2,3).forEach(num_test -> {
14             DatosEjercicio3.iniDatos("ficheros/Ejercicio3DatosEntrada"+num_test+".txt");
15             Ejercicio3BT.search();
16             Ejercicio3BT.getSoluciones().forEach(s -> String2.toConsole("Solucion obtenida:
17 %s\n", s));
18             TestsPI5.Line("*");
19         });
20     }
21 }
22
```

RESULTADOS GRAFOS VIRTUALES

```
Investigadores: [Investigador[nombre=INV1, capacidad=6, especialidad=0], Investigador[nombre=INV2, capacidad=3, especialidad=1],  
Investigador[nombre=INV3, capacidad=8, especialidad=2]]  
trabajos = {  
    Trabajo[nombre=T01, calidad=5, diasRequeridos=[6, 0, 0]]  
    Trabajo[nombre=T02, calidad=10, diasRequeridos=[0, 3, 8]]  
}  
especialidades = {  
    0  
    1  
    2  
}  
=====  
Voraz no obtuvo solucion  


---

INV1: [6, 0]  
INV2: [0, 3]  
INV3: [0, 8]  
Solucion A*: calidades: 15.0  


---

INV1: [6, 0]  
INV2: [0, 3]  
INV3: [0, 8]  
Solucion PDR: calidades: 15.0  


---

INV1: [6, 0]  
INV2: [0, 3]  
INV3: [0, 8]  
Solucion BT: calidades: 15.0  
*****  
Investigadores: [Investigador[nombre=INV1, capacidad=10, especialidad=0], Investigador[nombre=INV2, capacidad=5, especialidad=1],  
Investigador[nombre=INV3, capacidad=8, especialidad=2], Investigador[nombre=INV4, capacidad=2, especialidad=0],  
Investigador[nombre=INV5, capacidad=5, especialidad=3]]  
trabajos = {  
    Trabajo[nombre=T01, calidad=7, diasRequeridos=[2, 0, 5, 0]]  
    Trabajo[nombre=T02, calidad=9, diasRequeridos=[8, 4, 3, 0]]  
    Trabajo[nombre=T03, calidad=5, diasRequeridos=[2, 0, 0, 7]]  
}  
especialidades = {  
    0  
    1  
    2  
    3  
}  
=====  
Voraz no obtuvo solucion  


---

INV1: [1, 8, 0]  
INV2: [0, 4, 0]  
INV3: [5, 3, 0]  
INV4: [1, 0, 1]  
INV5: [0, 0, 0]  
Solucion A*: calidades: 16.0  


---

INV1: [0, 8, 0]  
INV2: [0, 4, 0]  
INV3: [5, 3, 0]  
INV4: [2, 0, 0]  
INV5: [0, 0, 0]  
Solucion PDR: calidades: 16.0  


---

INV1: [0, 8, 0]  
INV2: [0, 4, 0]  
INV3: [5, 3, 0]  
INV4: [2, 0, 0]  
INV5: [0, 0, 0]  
Solucion BT: calidades: 16.0  
*****  
Investigadores: [Investigador[nombre=INV1, capacidad=1, especialidad=2], Investigador[nombre=INV2, capacidad=10, especialidad=1],  
Investigador[nombre=INV3, capacidad=3, especialidad=0], Investigador[nombre=INV4, capacidad=4, especialidad=0],  
Investigador[nombre=INV5, capacidad=10, especialidad=3], Investigador[nombre=INV6, capacidad=4, especialidad=3],  
Investigador[nombre=INV7, capacidad=1, especialidad=2], Investigador[nombre=INV8, capacidad=30, especialidad=3]]  
trabajos = {  
    Trabajo[nombre=T01, calidad=8, diasRequeridos=[2, 0, 2, 0]]  
    Trabajo[nombre=T02, calidad=5, diasRequeridos=[8, 5, 4, 2]]  
    Trabajo[nombre=T03, calidad=8, diasRequeridos=[0, 5, 0, 15]]  
    Trabajo[nombre=T04, calidad=5, diasRequeridos=[0, 7, 8, 5]]  
    Trabajo[nombre=T05, calidad=9, diasRequeridos=[5, 5, 0, 2]]  
}  
especialidades = {  
    2  
    1  
    0  
    3  
}  
=====  
Voraz no obtuvo solucion
```

RESULTADOS MANUAL

```
Investigadores: [Investigador[nombre=INV1, capacidad=6, especialidad=0], Investigador[nombre=INV2, capacidad=3, especialidad=1], Investigador[nombre=INV3, capacidad=8, especialidad=2]]
```

```
trabajos = {
```

```
    Trabajo[nombre=T01, calidad=5, diasRequeridos=[6, 0, 0]]
```

```
    Trabajo[nombre=T02, calidad=10, diasRequeridos=[0, 3, 8]]
```

```
}
```

```
especialidades = {
```

```
    0
```

```
    1
```

```
    2
```

```
}
```

```
[[6, 0, 0], [0, 3, 8]]
```

```
INV1: [6, 0]
```

```
INV2: [0, 3]
```

```
INV3: [0, 8]
```

```
Solucion obtenida: calidades: 15.0
```

```
*****
```

```
Investigadores: [Investigador[nombre=INV1, capacidad=10, especialidad=0], Investigador[nombre=INV2, capacidad=5, especialidad=1], Investigador[nombre=INV3, capacidad=8, especialidad=2], Investigador[nombre=INV4, capacidad=2, especialidad=0], Investigador[nombre=INV5, capacidad=5, especialidad=3]]
```

```
trabajos = {
```

```
    Trabajo[nombre=T01, calidad=7, diasRequeridos=[2, 0, 5, 0]]
```

```
    Trabajo[nombre=T02, calidad=9, diasRequeridos=[8, 4, 3, 0]]
```

```
    Trabajo[nombre=T03, calidad=5, diasRequeridos=[2, 0, 0, 7]]
```

```
}
```

```
especialidades = {
```

```
    0
```

```
    1
```

```
    2
```

```
    3
```

```
}
```

```
[[0, 0, 5, 2, 0], [8, 4, 3, 0, 0], [0, 0, 0, 0, 0]]
```

```
INV1: [0, 8, 0]
```

```
INV2: [0, 4, 0]
```

```
INV3: [5, 3, 0]
```

```
INV4: [2, 0, 0]
```

```
INV5: [0, 0, 0]
```

```
Solucion obtenida: calidades: 16.0
```

```
*****
```

```
Investigadores: [Investigador[nombre=INV1, capacidad=1, especialidad=2], Investigador[nombre=INV2, capacidad=10, especialidad=1], Investigador[nombre=INV3, capacidad=3, especialidad=0], Investigador[nombre=INV4, capacidad=4, especialidad=0], Investigador[nombre=INV5, capacidad=10, especialidad=3], Investigador[nombre=INV6, capacidad=4, especialidad=3], Investigador[nombre=INV7, capacidad=1, especialidad=2], Investigador[nombre=INV8, capacidad=30, especialidad=3]]
```

```
trabajos = {
```

```
    Trabajo[nombre=T01, calidad=8, diasRequeridos=[2, 0, 2, 0]]
```

```
    Trabajo[nombre=T02, calidad=5, diasRequeridos=[8, 5, 4, 2]]
```

```
    Trabajo[nombre=T03, calidad=8, diasRequeridos=[0, 5, 0, 15]]
```

```
    Trabajo[nombre=T04, calidad=5, diasRequeridos=[0, 7, 8, 5]]
```

```
    Trabajo[nombre=T05, calidad=9, diasRequeridos=[5, 5, 0, 2]]
```

```
}
```

```
especialidades = {
```

```
    2
```

```
    1
```

```
    0
```

```
    3
```

```
}
```

EJERCICIO 4

```
1 package ejercicios.ejercicio4;
2
3 import java.util.ArrayList;
11
12 public record Ejercicio4Vertex(Integer cliente, Set<Integer> pendientes, List<Integer>
    visitados, Integer kms)
13     implements VirtualVertex <Ejercicio4Vertex,Ejercicio4Edge,Integer> {
14
15     public static Ejercicio4Vertex of(Integer c, Set<Integer> p, List<Integer> v, Integer k)
16     {
17         return new Ejercicio4Vertex(c, p, v, k);
18     }
19
20     public static Ejercicio4Vertex initial() {
21         Set<Integer> pend = new HashSet<>();
22         List<Integer> clientes = new ArrayList<>(DatosEjercicio4.getClientes().stream().map
23             (c->c.id()).toList());
24         List<Integer> visitados = new ArrayList<>();
25         Integer km = 0;
26
27         for(int i =0; i<DatosEjercicio4.getN();i++) {
28             pend.add(clientes.get(i));
29         }
30         return of(0,pend,visitados,km);
31     }
32
33     public static Predicate<Ejercicio4Vertex> goal(){
34         return v->v.visitados().size() == DatosEjercicio4.getN();
35     }
36
37     public static Predicate<Ejercicio4Vertex> goalHasSolution(){
38         return v->v.pendientes().isEmpty();
39     }
40
41     @Override
42     public List<Integer> actions() {
43         List<Integer> alternativas = new ArrayList<>();
44         List<Integer> pend = new ArrayList<>(pendientes);
45         if(cliente<DatosEjercicio4.getN()) {
46             if (!pendientes.isEmpty()) {
47                 if(pend.size()==1) {
48                     if(pend.get(pend.size()-1)==0) {
49                         alternativas.add(pend.get(pend.size()-1));
50                     }
51                 } else {
52                     for(int i =0; i<pendientes.size();i++) {
53                         if(DatosEjercicio4.existeArista(cliente, pend.get(i))) {
54                             alternativas.add(pend.get(i));
55                         }
56                     }
57                 }
58             }
59         }
60         return alternativas;
61     }
62
63     @Override
64     public Ejercicio4Vertex neighbor(Integer a) {
65
66         Set<Integer> newPend = new HashSet<>(pendientes);
67         List<Integer> newVisi = new ArrayList<>(visitados);
68         Integer newKms = kms;
```

```
67     newPend.remove(a);
68     newVisi.add(a);
69
70     if (newVisi.size() > 1) {
71         Integer ultimo = newVisi.get(newVisi.size() - 2);
72         if (DatosEjercicio4.existeArista(ultimo, a)) {
73             Double dist = DatosEjercicio4.getKm(ultimo, a);
74             newKms += dist.intValue();
75         }
76     }
77 }
78
79     return of(a,newPend,newVisi,newKms);
80 }
81
82 @Override
83 public Ejercicio4Edge edge(Integer a) {
84     return Ejercicio4Edge.of(this, neighbor(a), a);
85 }
86
87 public Ejercicio4Edge greedyEdge() {
88     return null;
89 }
90 }
91 }
```

```
1 package ejercicios.ejercicio4;
2
3 import _datos.DatosEjercicio4;
4
5
6 public record Ejercicio4Edge(Ejercicio4Vertex source, Ejercicio4Vertex target, Integer
7     action, Double weight)
8     implements SimpleEdgeAction <Ejercicio4Vertex, Integer> {
9
10    public static Ejercicio4Edge of(Ejercicio4Vertex s, Ejercicio4Vertex t, Integer a) {
11        return new Ejercicio4Edge(s, t, a, DatosEjercicio4.getBeneficio(t.cliente())-t.kms
12        ());
13    }
14
15    @Override
16    public String toString() {
17        return String.format("%d; %.1f", action, weight);
18    }
19
20}
```

```
1 package ejercicios.ejercicio4;
2
3 import java.util.function.Predicate;
4
5
6 public class Ejercicio4Heuristic {
7
8     public static Double heuristic(Ejercicio4Vertex v1, Predicate<Ejercicio4Vertex> goal,
9                                     Ejercicio4Vertex v2) {
10
11         return IntStream.range(0, DatosEjercicio4.getN())
12             .filter(i ->!v1.visitados().contains(i) && v1.pendientes().contains(i) &&
13                   v2!=null)
14             .mapToDouble(i -> DatosEjercicio4.getBeneficio(i) - (v1.kms() +
15                   DatosEjercicio4.getKm(v1.cliente(),i)))
16             .sum();
17
18     }
19 }
```

```
1 package tests;
2
3 import java.util.List;
10
11 public class TestEjercicio4 {
12
13     public static void main(String[] args) {
14
15         List.of(1,2).forEach(num_test -> {
16             TestsPI5.iniTest2("Ejercicio4DatosEntrada", num_test,
17             DatosEjercicio4::iniDatos);
18
19             // TODO Defina en el tipo vertice un m. factorial para el vertice inicial
20             // TODO Defina en el tipo vertice un m. static / Predicate para los vertices
21             finales
22                 TestsPI5.tests(
23                     Ejercicio4Vertex.initial(),           // Vertice Inicial
24                     Ejercicio4Vertex.goal(),            // Predicado para un vertice final
25                     GraphsPI5::multisetBuilder,          // Referencia al Builder del grafo
26                     Ejercicio4Vertex::greedyEdge,        // Referencia a la Funcion para la arista
27                     voraz
28                     SolucionEjercicio4::of);           // Referencia al metodo factorial para la
29
30     }
31 }
```

```
1 package ejercicios.ejercicio4.manual;
2
3 import java.util.ArrayList;
10
11 public record Ejercicio4Problem(Integer cliente, Set<Integer> pendientes, List<Integer>
12     visitados, Integer kms) {
13
14     public static Ejercicio4Problem of(Integer c, Set<Integer> p, List<Integer> v, Integer
15         k) {
16         return new Ejercicio4Problem(c, p, v, k);
17     }
18
19     public static Ejercicio4Problem initial() {
20         Set<Integer> pend = new HashSet<>();
21         List<Integer> clientes = new ArrayList<>(DatosEjercicio4.getClientes().stream().map
22             (c->c.id()).toList());
23         List<Integer> visitados = new ArrayList<>();
24         Integer km = 0;
25
26         for(int i =0; i<DatosEjercicio4.getN();i++) {
27             pend.add(clientes.get(i));
28         }
29         return of(0,pend,visitados,km);
30     }
31
32     public List<Integer> actions() {
33         List<Integer> alternativas = new ArrayList<>();
34         List<Integer> pend = new ArrayList<>(pendientes);
35         if(cliente<DatosEjercicio4.getN()) {
36             if (!pendientes.isEmpty()) {
37                 if(pend.size()==1) {
38                     if(pend.get(pend.size()-1)==0) {
39                         alternativas.add(pend.get(pend.size()-1));
40                     }
41                 } else {
42                     for(int i =0; i<pendientes.size();i++) {
43                         if(DatosEjercicio4.existeArista(cliente, pend.get(i))) {
44                             alternativas.add(pend.get(i));
45                         }
46                     }
47                 }
48             }
49         }
50         return alternativas;
51     }
52
53     public Ejercicio4Problem neighbor(Integer a) {
54
55         Set<Integer> newPend = new HashSet<>(pendientes);
56         List<Integer> newVisi = new ArrayList<>(visitados);
57         Integer newKms = kms;
58
59         newPend.remove(a);
60         newVisi.add(a);
61
62         if (newVisi.size() > 1) {
63             Integer ultimo = newVisi.get(newVisi.size() - 2);
64             if (DatosEjercicio4.existeArista(ultimo, a)) {
65                 Double dist = DatosEjercicio4.getKm(ultimo, a);
66                 newKms += dist.intValue();
67             }
68         }
69     }
70 }
```

```
66         }
67
68     return of(a,newPend,newVisi,newKms);
69 }
70
71 public Double heuristic() {
72     return IntStream.range(0, DatosEjercicio4.getN())
73         .filter(i ->!visitados.contains(i) && pendientes.contains(i))
74         .mapToDouble(i -> DatosEjercicio4.getBeneficio(i) - kms)
75         .sum();
76 }
77
78 }
79
```

```
1 package ejercicios.ejercicio4.manual;
2
3 import java.util.List;
4
5 public class Ejercicio4State {
6
7     Ejercicio4Problem actual;
8     Double acumulado;
9     List<Integer> acciones;
10    List<Ejercicio4Problem> anteriores;
11
12    private Ejercicio4State(Ejercicio4Problem p, Double a, List<Integer> ls1,
13     List<Ejercicio4Problem> ls2) {
14        actual = p;
15        acumulado = a;
16        acciones = ls1;
17        anteriores = ls2;
18    }
19
20    public static Ejercicio4State initial() {
21        Ejercicio4Problem pi = Ejercicio4Problem.initial();
22        return of(pi,0.,List2.empty(),List2.empty());
23    }
24
25    public static Ejercicio4State of(Ejercicio4Problem prob, Double acum,
26     List<Integer> lsa,
27     List<Ejercicio4Problem> lsp) {
28        return new Ejercicio4State(prob, acum, lsa, lsp);
29    }
30
31    public void forward(Integer a) {
32        acumulado += DatosEjercicio4.getBeneficio(actual.cliente()) - actual.kms();
33        acciones.add(a);
34        anteriores.add(actual);
35        actual = actual.neighbor(a);
36    }
37
38    public void back() {
39        int prev = acciones.size()-1;
40        var last = anteriores.get(prev);
41        // guardar antes las variables
42        acumulado -= DatosEjercicio4.getBeneficio(last.cliente()) - last.kms();
43        acciones.remove(prev);
44        anteriores.remove(prev);
45        actual = last;
46    }
47
48    public List<Integer> alternativas() {
49        return actual.actions();
50    }
51
52    public Double cota(Integer a) {
53        Double weight = DatosEjercicio4.getBeneficio(actual.cliente()).doubleValue() -
54            actual.kms();
55        return acumulado + weight + actual.neighbor(a).heuristic();
56    }
57
58    public Boolean esSolucion() {
59        return actual.visitados().size() == DatosEjercicio4.getN();
60    }
61
62    public Boolean esTerminal() {
63        return actual.pendientes().isEmpty();
```

```
64      }
65
66  public SolucionEjercicio4 getSolucion() {
67      return SolucionEjercicio4.of(acciones);
68  }
69}
70
```

```
1 package ejercicios.ejercicio4.manual;
2
3 import java.util.HashSet;
4
5
6 public class Ejercicio4BT {
7
8     private static Double mejorValor;
9
10    private static Ejercicio4State estado;
11
12    private static Set<SolucionEjercicio4> soluciones;
13
14    public static void search() {
15        soluciones = new HashSet<>();
16        mejorValor = Double.MIN_VALUE;
17        estado = Ejercicio4State.initial();
18        bt_search();
19    }
20
21    private static void bt_search() {
22        if (estado.esSolucion()) {
23            Double valorObtenido = estado.acumulado;
24            if (valorObtenido > mejorValor) {
25                mejorValor=valorObtenido;
26                soluciones.add(estado.getSolucion());
27            }
28        } else if (!estado.esTerminal()) {
29            for (Integer a: estado.alternativas()) {
30                if (estado.cota(a) > mejorValor) {
31                    estado.forward(a);
32                    bt_search();
33                    estado.back();
34                }
35            }
36        }
37    }
38
39    public static Set<SolucionEjercicio4> getSoluciones() {
40        return soluciones;
41    }
42
43 }
44
```

```
1 package tests.manual;
2
3 import java.util.List;
9
10 public class TestEjercicio4BT {
11
12     public static void main(String[] args) {
13         List.of(1,2).forEach(num_test -> {
14             DatosEjercicio4.iniDatos("Ejercicio4DatosEntrada"+num_test);
15             Ejercicio4BT.search();
16             Ejercicio4BT.getSoluciones().forEach(s -> String2.toConsole("Solucion obtenida:
17 %s\n", s));
18             TestsPI5.Line("*");
19         });
20     }
21
22 }
23
```

RESULTADOS GRAFOS VIRTUALES

Archivo Ejercicio4DatosEntrada1.txt

Datos de entrada: ([Cliente[id=0, beneficio=0.0], Cliente[id=1, beneficio=400.0], Cliente[id=2, beneficio=300.0], Cliente[id=3, beneficio=200.0], Cliente[id=4, beneficio=100.0]], [Conexion[cliente1=0, cliente2=1, kms=1.0]={Cliente[id=0, beneficio=0.0],Cliente[id=1, beneficio=400.0]}, Conexion[cliente1=0, cliente2=3, kms=100.0]={Cliente[id=0, beneficio=0.0],Cliente[id=3, beneficio=200.0]}, Conexion[cliente1=1, cliente2=2, kms=1.0]={Cliente[id=1, beneficio=400.0],Cliente[id=2, beneficio=300.0]}, Conexion[cliente1=1, cliente2=3, kms=100.0]={Cliente[id=1, beneficio=400.0],Cliente[id=3, beneficio=200.0]}, Conexion[cliente1=2, cliente2=3, kms=1.0]={Cliente[id=2, beneficio=300.0],Cliente[id=3, beneficio=200.0]}, Conexion[cliente1=3, cliente2=4, kms=1.0]={Cliente[id=3, beneficio=200.0],Cliente[id=4, beneficio=100.0]}, Conexion[cliente1=2, cliente2=4, kms=100.0]={Cliente[id=2, beneficio=300.0],Cliente[id=4, beneficio=100.0]}, Conexion[cliente1=0, cliente2=4, kms=5.0]={Cliente[id=0, beneficio=0.0],Cliente[id=4, beneficio=100.0]})

Voraz no obtuvo solucion

[1, 2, 3, 4, 0]

Solucion A*: SolucionEjercicio4 [kms=9.0, beneficio=981.0, clientes=[Cliente[id=1, beneficio=400.0], Cliente[id=2, beneficio=300.0], Cliente[id=3, beneficio=200.0], Cliente[id=4, beneficio=100.0], Cliente[id=0, beneficio=0.0]]]

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

[1, 2, 3, 4, 0]

Solucion PDR: SolucionEjercicio4 [kms=9.0, beneficio=981.0, clientes=[Cliente[id=1, beneficio=400.0], Cliente[id=2, beneficio=300.0], Cliente[id=3, beneficio=200.0], Cliente[id=4, beneficio=100.0], Cliente[id=0, beneficio=0.0]]]

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

[1, 2, 3, 4, 0]

Solucion BT: SolucionEjercicio4 [kms=9.0, beneficio=981.0, clientes=[Cliente[id=1, beneficio=400.0], Cliente[id=2, beneficio=300.0], Cliente[id=3, beneficio=200.0], Cliente[id=4, beneficio=100.0], Cliente[id=0, beneficio=0.0]]]

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

Archivo Ejercicio4DatosEntrada2.txt

Datos de entrada: ([Cliente[id=0, beneficio=0.0], Cliente[id=1, beneficio=100.0], Cliente[id=2, beneficio=200.0], Cliente[id=3, beneficio=300.0], Cliente[id=4, beneficio=200.0], Cliente[id=5, beneficio=300.0], Cliente[id=6, beneficio=200.0], Cliente[id=7, beneficio=200.0]], [Conexion[cliente1=0, cliente2=1, kms=2.0]={Cliente[id=0, beneficio=0.0],Cliente[id=1, beneficio=100.0]}, Conexion[cliente1=0, cliente2=2, kms=1.0]={Cliente[id=0, beneficio=0.0],Cliente[id=2, beneficio=200.0]}, Conexion[cliente1=0, cliente2=4, kms=1.0]={Cliente[id=0, beneficio=0.0],Cliente[id=4, beneficio=200.0]}, Conexion[cliente1=2, cliente2=7, kms=3.0]={Cliente[id=2, beneficio=200.0],Cliente[id=7, beneficio=200.0]}, Conexion[cliente1=4, cliente2=7, kms=1.0]={Cliente[id=4, beneficio=200.0],Cliente[id=7, beneficio=200.0]}, Conexion[cliente1=1, cliente2=6, kms=1.0]={Cliente[id=1, beneficio=100.0],Cliente[id=6, beneficio=200.0]}, Conexion[cliente1=5, cliente2=6, kms=3.0]={Cliente[id=5, beneficio=300.0],Cliente[id=6, beneficio=200.0]}, Conexion[cliente1=2, cliente2=5, kms=1.0]={Cliente[id=2, beneficio=200.0],Cliente[id=5, beneficio=300.0]}, Conexion[cliente1=3, cliente2=7, kms=1.0]={Cliente[id=3, beneficio=300.0],Cliente[id=7, beneficio=200.0]}, Conexion[cliente1=3, cliente2=1, kms=3.0]={Cliente[id=3, beneficio=300.0],Cliente[id=1, beneficio=100.0]}, Conexion[cliente1=3, cliente2=5, kms=1.0]={Cliente[id=3, beneficio=300.0],Cliente[id=5, beneficio=300.0]}, Conexion[cliente1=4, cliente2=6, kms=1.0]={Cliente[id=4, beneficio=200.0],Cliente[id=6, beneficio=200.0]}, Conexion[cliente1=2, cliente2=1, kms=1.0]={Cliente[id=2, beneficio=200.0],Cliente[id=1, beneficio=100.0]})

Voraz no obtuvo solucion

[2, 5, 3, 7, 4, 6, 1, 0]

Solucion A*: SolucionEjercicio4 [kms=9.0, beneficio=1463.0, clientes=[Cliente[id=2, beneficio=200.0], Cliente[id=5, beneficio=300.0], Cliente[id=3, beneficio=300.0], Cliente[id=7, beneficio=200.0], Cliente[id=4, beneficio=200.0], Cliente[id=6, beneficio=200.0], Cliente[id=1, beneficio=100.0], Cliente[id=0, beneficio=0.0]]]

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

[1, 2, 5, 3, 7, 4, 6, 0]

Solucion PDR: SolucionEjercicio4 [kms=8.0, beneficio=1465.0, clientes=[Cliente[id=1, beneficio=100.0], Cliente[id=2, beneficio=200.0], Cliente[id=5, beneficio=300.0], Cliente[id=3, beneficio=300.0], Cliente[id=7, beneficio=200.0], Cliente[id=4, beneficio=200.0], Cliente[id=6, beneficio=200.0], Cliente[id=0, beneficio=0.0]]]

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

[1, 2, 5, 3, 7, 4, 6, 0]

Solucion BT: SolucionEjercicio4 [kms=8.0, beneficio=1465.0, clientes=[Cliente[id=1, beneficio=100.0], Cliente[id=2, beneficio=200.0], Cliente[id=5, beneficio=300.0], Cliente[id=3, beneficio=300.0], Cliente[id=7, beneficio=200.0], Cliente[id=4, beneficio=200.0], Cliente[id=6, beneficio=200.0], Cliente[id=0, beneficio=0.0]]]

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

RESULTADOS MANUAL

Archivo Ejercicio4DatosEntrada1.txt

Datos de entrada: ([Cliente[id=0, beneficio=0.0], Cliente[id=1, beneficio=400.0], Cliente[id=2, beneficio=300.0], Cliente[id=3, beneficio=200.0], Cliente[id=4, beneficio=100.0]], [Conexion[cliente1=0, cliente2=1, kms=1.0]={Cliente[id=0, beneficio=0.0],Cliente[id=1, beneficio=400.0]}, Conexion[cliente1=0, cliente2=3, kms=100.0]={Cliente[id=0, beneficio=0.0],Cliente[id=3, beneficio=200.0]}, Conexion[cliente1=1, cliente2=2, kms=1.0]={Cliente[id=1, beneficio=400.0],Cliente[id=2, beneficio=300.0]}, Conexion[cliente1=1, cliente2=3, kms=100.0]={Cliente[id=1, beneficio=400.0],Cliente[id=3, beneficio=200.0]}, Conexion[cliente1=2, cliente2=3, kms=1.0]={Cliente[id=2, beneficio=300.0],Cliente[id=3, beneficio=200.0]}, Conexion[cliente1=3, cliente2=4, kms=1.0]={Cliente[id=3, beneficio=200.0],Cliente[id=4, beneficio=100.0]}, Conexion[cliente1=2, cliente2=4, kms=100.0]={Cliente[id=2, beneficio=300.0],Cliente[id=4, beneficio=100.0]}, Conexion[cliente1=0, cliente2=4, kms=5.0]={Cliente[id=0, beneficio=0.0],Cliente[id=4, beneficio=100.0]}])
[1, 2, 3, 4, 0]

Solucion obtenida: SolucionEjercicio4 [kms=9.0, beneficio=981.0, clientes=[Cliente[id=1, beneficio=400.0], Cliente[id=2, beneficio=300.0], Cliente[id=3, beneficio=200.0], Cliente[id=4, beneficio=100.0], Cliente[id=0, beneficio=0.0]]]

Archivo Ejercicio4DatosEntrada2.txt

Datos de entrada: ([Cliente[id=0, beneficio=0.0], Cliente[id=1, beneficio=100.0], Cliente[id=2, beneficio=200.0], Cliente[id=3, beneficio=300.0], Cliente[id=4, beneficio=200.0], Cliente[id=5, beneficio=300.0], Cliente[id=6, beneficio=200.0], Cliente[id=7, beneficio=200.0]], [Conexion[cliente1=0, cliente2=1, kms=2.0]={Cliente[id=0, beneficio=0.0],Cliente[id=1, beneficio=100.0]}, Conexion[cliente1=0, cliente2=2, kms=1.0]={Cliente[id=0, beneficio=0.0],Cliente[id=2, beneficio=200.0]}, Conexion[cliente1=0, cliente2=4, kms=1.0]={Cliente[id=0, beneficio=0.0],Cliente[id=4, beneficio=200.0]}, Conexion[cliente1=2, cliente2=7, kms=3.0]={Cliente[id=2, beneficio=200.0],Cliente[id=7, beneficio=200.0]}, Conexion[cliente1=4, cliente2=7, kms=1.0]={Cliente[id=4, beneficio=200.0],Cliente[id=7, beneficio=200.0]}, Conexion[cliente1=1, cliente2=6, kms=1.0]={Cliente[id=1, beneficio=100.0],Cliente[id=6, beneficio=200.0]}, Conexion[cliente1=5, cliente2=6, kms=3.0]={Cliente[id=5, beneficio=300.0],Cliente[id=6, beneficio=200.0]}, Conexion[cliente1=2, cliente2=5, kms=1.0]={Cliente[id=2, beneficio=200.0],Cliente[id=5, beneficio=300.0]}, Conexion[cliente1=3, cliente2=7, kms=1.0]={Cliente[id=3, beneficio=300.0],Cliente[id=7, beneficio=200.0]}, Conexion[cliente1=3, cliente2=1, kms=3.0]={Cliente[id=3, beneficio=300.0],Cliente[id=1, beneficio=100.0]}, Conexion[cliente1=3, cliente2=5, kms=1.0]={Cliente[id=3, beneficio=300.0],Cliente[id=5, beneficio=300.0]}, Conexion[cliente1=4, cliente2=6, kms=1.0]={Cliente[id=4, beneficio=200.0],Cliente[id=6, beneficio=200.0]}, Conexion[cliente1=2, cliente2=1, kms=1.0]={Cliente[id=2, beneficio=200.0],Cliente[id=1, beneficio=100.0]}])
[1, 2, 5, 3, 7, 4, 6, 0]

Solucion obtenida: SolucionEjercicio4 [kms=8.0, beneficio=1465.0, clientes=[Cliente[id=1, beneficio=100.0], Cliente[id=2, beneficio=200.0], Cliente[id=5, beneficio=300.0], Cliente[id=3, beneficio=300.0], Cliente[id=7, beneficio=200.0], Cliente[id=4, beneficio=200.0], Cliente[id=6, beneficio=200.0], Cliente[id=0, beneficio=0.0]]]
