

Práctica Individual 2

<Problema Número 3>

Análisis y Diseño de Datos y Algoritmos
Grado de Ingeniería Informática del Software
Curso 2º

Francisco Javier Huertas Vera (javiihv93@gmail.com)

Tutor: Fernando de la Rosa

Índice

1	Enunciado y Cuestiones a resolver	3
2	Seguimiento.....	4
2.1	Control de seguimiento y tutorías	4
2.2	Portafolio	4
3	Implementación.....	5
4	Test y Pruebas.....	9

1 Enunciado y Cuestiones a resolver

Enunciado:

La nave de la flota intergaláctica Enterprise se tiene que preparar para un viaje por el espacio profundo. Una de las tareas más importantes es seleccionar las armas que han de llevar en la nave. Ésta ha de contener todas las características requeridas para la correcta protección de la nave y, además, ha de pesar lo menos posible. Para llenar la nave, es necesario considerar que hay una lista de características requeridas que deben cubrirse por las armas seleccionadas. Por ejemplo, a continuación, se muestra una lista de características requeridas para un escenario concreto:

Requisito-1	Requisito-2	Requisito-3	Requisito-4
Láser	Paralizante	Cuerpo-Cuerpo	Ligera

Para llenar la bodega de la nave, se dispone de un hangar con las distintas armas que están disponibles, dichas armas se describen por sus características, su peso y el daño que producen. Un arma del hangar puede responder a más de una necesidad, por ejemplo, si se necesita un arma láser ligera y un arma láser, con un arma láser ligera sería suficiente. A continuación, se muestra un ejemplo de hangar donde se dispone de las siguientes armas:

Nombre:	P-90	P-90-2	Disruptor Kull	Zat	Orizouk
Peso:	150g	250g	250g	220g	100g
Características:	Láser, Ligera	Láser, Ligera	Paralizante, Ligera	Láser, Paralizante, Cuerpo-Cuerpo	Torpedo
Daño:	750	650	500	350	100

Una solución podría ser las armas P-90 -que cumple la primera y la cuarta características requeridas de la lista previamente indicada- y Zat – que cumple la primera, segunda y tercera características requeridas de la lista previamente indicada-. El peso de dicha solución sería $150 + 220 = 370$ g. Otra posible solución sería utilizar Disruptor Kull, Zat e OriZouk cuyo peso sería 570 g. El objetivo es minimizar el peso que llevamos en la nave, por lo que la primera solución será la elegida. Modele y resuelva este problema utilizando grafos virtuales de manera que, de cada vértice salgan dos aristas, una indicando que se coge cierta arma del hangar y otra indicando que dicho arma no se coge. Para ello considere que los objetos del hangar (clase ObjetoHangar) tienen identificación (String), peso (Float), daño (Integer), y una lista de características (List). Las necesidades del viaje se especificarán como una lista de

características. Cada arma tendrá como propiedad una lista de características, por lo tanto, las necesidades son listas de listas de características (List).

Preguntas

1. Justifique el tipo de grafo virtual que ha de utilizar y diseñe la clase de sus vértices.
2. Indique justificadamente cómo modelará los pesos de los elementos del grafo.
3. Implemente todo lo necesario para resolver el problema mediante el algoritmo de AStar. Puede utilizar el código provisto en el paquete `us.lsi.astar.viajeintergalactico`.
4. Complete la clase `TestViajeInterestelar` de modo que permita lanzar, al menos, el ejemplo indicado.

2 Seguimiento

2.1 Control de seguimiento y tutorías

El alumno debe listar y detallar las visitas a tutoría

Fecha	Profesor	Detalles y dudas resueltas
04/10/2016	X	- Problemas de comprensión de enunciado, etc.

2.2 Portafolio

Encontré problemas con el peso de los vértices, ya que creía que el peso sería el peso del arma que se coge en cada vértice y si no se cogía ningún arma sería 0. En la clase de seguimiento el profesor me resolvió esta duda, y el peso de los vértices sería 0, y lo que tendrá peso serán las aristas, que será el peso del arma que se ha cogido o 0 en el caso de que no se coja el arma.

3 Implementación

Clases y métodos implementados.

1. Justifique el tipo de grafo virtual que ha de utilizar y diseñe la clase de sus vértices.

El grado que utilizaremos será un grafo virtual.

Los vértices serán del tipo EstadoNave, que cada vértice nos informará de las armas que se han seleccionado hasta el momento.

La clase EstadoNave será la siguiente:

```
public class EstadoNave implements VirtualVertex<EstadoNave,
SimpleEdge<EstadoNave>> {

    static List<ObjetoHangar> hangar;

    private List<ObjetoHangar> armas_seleccionadas;
    private int arma_i;

    public EstadoNave() {
        //TODO
        super();
        this.armas_seleccionadas= new ArrayList<ObjetoHangar>();
        this.arma_i = 0;
    }

    public EstadoNave(List<ObjetoHangar> armas_seleccionadas, int arma_i,
List<ObjetoHangar> hangar) {
        // TODO
        this.arma_i=arma_i;

        this.armas_seleccionadas=armas_seleccionadas;
        this.hangar = hangar;
    }

    @Override
    public boolean isValid() {
        boolean res = false;
        if(this.arma_i>=0 || this.arma_i<=this.hangar.size()){
            res = true;
        }
        return res;
    }

    @Override
    public Set<EstadoNave> getNeighborListof() {
        //conjuntoVecinos
        Set<EstadoNave> ret = new HashSet<EstadoNave>();
        Integer arma1 = this.arma_i;
```

```

        List<ObjetoHangar> l = new
ArrayList<ObjetoHangar>(this.armas_seleccionadas);
        //CogeArma
        l.add(hangar.get(arma_i));
        EstadoNave cogeArma = new EstadoNave(l, arma1+1, this.hangar);
        //Float peso1 = hangar.get(arma_i).getPeso();
        //NoCogeArma
        List<ObjetoHangar> l2 = new
ArrayList<ObjetoHangar>(this.armas_seleccionadas);
        EstadoNave noCogeArma = new EstadoNave(l2, arma1+1, this.hangar);
        //AñadeVecinos
        ret.add(cogeArma);
        ret.add(noCogeArma);

        return ret;
    }

    @Override
    public Set<SimpleEdge<EstadoNave>> edgesOf() {

        return getNeighborListOf().stream()
                .map(v -> SimpleEdge.create(this, v))
                .collect(Collectors.toSet());
    }

    @Override
    public boolean isNeighbor(EstadoNave e) {
        boolean res = false;
        if(this.getNeighborListOf().contains(e)){
            res = true;
        }
        return res;
    }

    public List<ObjetoHangar> getArmas_seleccionadas() {
        // TODO Auto-generated method stub
        return this.armas_seleccionadas;
    }

    public int getArma_i(){
        return this.arma_i;
    }

    public double getPeso(){
        Double res = 0.;
        for(ObjetoHangar o : this.armas_seleccionadas){
            res += o.getPeso();
        }
        return res;
    }

    @Override
    public int hashCode() {
        final int prime = 31;

```

```

        int result = 1;
        result = prime * result + arma_i;
        result = prime * result + ((armas_seleccionadas == null) ? 0 :
armas_seleccionadas.hashCode());
        return result;
    }

    @Override
    public boolean equals(Object obj) {
        if (this == obj)
            return true;
        if (obj == null)
            return false;
        if (getClass() != obj.getClass())
            return false;
        EstadoNave other = (EstadoNave) obj;
        if (arma_i != other.arma_i)
            return false;
        if (armas_seleccionadas == null) {
            if (other.armas_seleccionadas != null)
                return false;
        } else if
(!armas_seleccionadas.equals(other.armas_seleccionadas))
            return false;
        return true;
    }

```

2. Indique justificadamente cómo modelará los pesos de los elementos del grafo.

Los vértices no tendrán peso. Las aristas si tendrán pesos y el peso se conseguirá restando el peso total de las armas seleccionadas del vértice fuente menos el peso total de las armas seleccionadas del vértice sumidero. El método es el siguiente:

-En este método calculamos el peso total de las armas seleccionadas:

```

public double getPeso(){
    Double res = 0.;
    for(ObjetoHangar o : this.armas_seleccionadas){
        res += o.getPeso();
    }
    return res;
}

```

-En este método calculamos el peso de la arista:

```

public double getEdgeWeight(SimpleEdge<EstadoNave> e) {
    double res = 0.;

    if(e.getSource().getArmas_seleccionadas().size()<e.getTarget().getArma
s_seleccionadas().size()){
        res = e.getTarget().getPeso()-e.getSource().getPeso();
    }
}

```

```

        return res;
    }

```

3. Implemente todo lo necesario para resolver el problema mediante el algoritmo de AStar. Puede utilizar el código provisto en el paquete `us.lsi.astar.viajeintergalactico`.

Las clases con las que modelaremos el grafo serán `EstadoNave` (añadida anteriormente), `GrafoNave` y `TestViajeIntelestelar` (añadida en el apartado 4). La clase `GrafoNave` es la siguiente:

```

public class GrafoNave extends UndirectedSimpleVirtualGraph<EstadoNave,
SimpleEdge<EstadoNave>>
    implements AStarGraph<EstadoNave, SimpleEdge<EstadoNave>> {

    public GrafoNave(EdgeFactory<EstadoNave, SimpleEdge<EstadoNave>>
edgeFactory) {
        super(edgeFactory);
    }
    public static GrafoNave create(EdgeFactory<EstadoNave,
SimpleEdge<EstadoNave>> edgeFactory) {
        return new GrafoNave(edgeFactory);
    }
    @Override
    public double getEdgeWeight(SimpleEdge<EstadoNave> e) {
        double res = 0.;

        if(e.getSource().getArmas_seleccionadas().size()<e.getTarget().getArma
s_seleccionadas().size()){
            res = e.getTarget().getPeso()-e.getSource().getPeso();
        }

        return res;
    }
    @Override
    public double getVertexWeight(EstadoNave vertex) {
        // TODO
        return 0.;
    }

    @Override
    public double getVertexWeight(EstadoNave vertex,
SimpleEdge<EstadoNave> edgeIn, SimpleEdge<EstadoNave> edgeOut) {
        // TODO
        return 0.;
    }
}

```



```

@Override
public double getWeightToEnd(EstadoNave actual, EstadoNave endVertex,
Function<EstadoNave, Double> goalDistance,
Set<EstadoNave> goalSet) {
    // TODO
    //utilizar la function del test goalDistance
    return 0.;
}
}

```

4 Test y Pruebas

- Adjuntar los test de prueba realizados para comprobar el funcionamiento de las clases y métodos desarrollados en cada apartado indicando distintos ejemplos de uso.
- Adjuntar las salidas obtenidas por los diferentes métodos en los casos de prueba

Adjuntamos ahora la clase TestViajeInterestelar en la que se han realizado las pruebas para comprobar el funcionamiento del problema:

```

public class TestViajeInterestelar {

    public static EstadoNave estadoInicial;
    public static void main(String[] args) {

        // TODO: Inicializa el problema hangar y requisitos
        List<String> requisitos= Lists.newArrayList("Laser", "Ligera",
"Paralizante",
                "Cuerpo-Cuerpo");
        List<ObjetoHangar> hangar = Lists.newArrayList(
                new ObjetoHangar("P-90",
Lists.newArrayList("Laser", "Ligera"), 150f, 750f),
                new ObjetoHangar("P-90-2",
Lists.newArrayList("Laser", "Ligera"), 250f, 650f),
                new ObjetoHangar("Disruptor kull",
Lists.newArrayList("Paralizante", "Ligera"),
                250f, 500f),
                new ObjetoHangar("Zat", Lists.newArrayList("Laser",
"Paralizante", "Cuerpo-Cuerpo"), 220f, 350f),
                new ObjetoHangar("OriZouk",
Lists.newArrayList("Torpedo"), 100f, 100f)
                );

        // TODO: Crear estado inicial vacio
        List<ObjetoHangar> l = new ArrayList<ObjetoHangar>();
        Integer i = 0;
        estadoInicial = new EstadoNave(l,i, hangar);
    }
}

```

```

// TODO: Generar function objetivo
Function<EstadoNave, Double> function = (e -> {
    if (requisitos.stream().allMatch(req ->
        e.getArmas_seleccionadas().stream().anyMatch(arma ->
            arma.caracteristicas.contains(req)))) {
        return 0.0;
    } else {
        return Double.MAX_VALUE;
    }
});

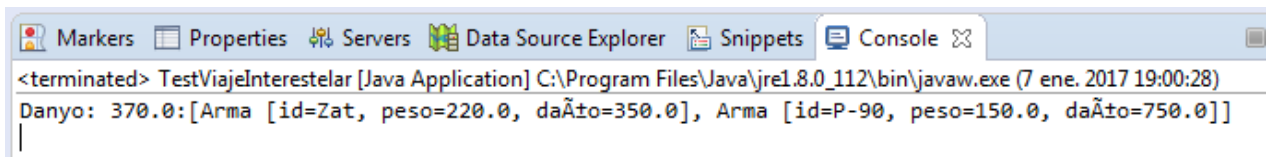
// TODO: Crear grafo Astart
AStarGraph<EstadoNave, SimpleEdge<EstadoNave>> graph =
GrafoNave.create(SimpleEdge::create);

// TODO: Crear Algoritmo AStar y le pasamos el grafo, el estado
inicial
// y el function
AStarAlgorithm<EstadoNave, SimpleEdge<EstadoNave>> alg =
Algoritmos.createAStar(graph, estadoInicial, function);

if (alg.getPath() == null) {
    System.out.println("No se encuentra solucion");
} else {
    System.out.println(
        "Danyo: " + alg.getPathLength() + ":" +
alg.getPath().getEndVertex().getArmas_seleccionadas());
}
}
}

```

La salida obtenida es la siguiente:



```

<terminated> TestViajeInterestelar [Java Application] C:\Program Files\Java\jre1.8.0_112\bin\javaw.exe (7 ene. 2017 19:00:28)
Danyo: 370.0:[Arma [id=Zat, peso=220.0, daÃ±o=350.0], Arma [id=P-90, peso=150.0, daÃ±o=750.0]]

```