PRÁCTICA 4 – Plantilla de algoritmo BaB para TSP

Estructuras de datos y Algoritmos II (EDA II). 2º Grado en Ingeniería Informática. Universidad de Almería

Version **curso.2022**

Plantilla de implementación en Java

A continuación tenéis un esquema o plantlla de la implementación del algoritmo ***branch-and-bound*** para el problema del TSP en un grafo representado por *un mapa de adyacencia*. Se trata de que completéis los bloques que faltan, marcados como comentarios dentro del código.

```
    // return the minimum value of the edges
    public double minimumEdgeValue() {
        double minimum = Double.MAX_VALUE;
        // Devuelve el menor valor de arista del grafo
        // Dos bucles 'for' anidados


        return minimum;
    }

    // TSP - BaB - Best-First
    public ArrayList<Vertex> TSPBaB(Vertex source) {
        TreeMap<Vertex, Double> neighborMap = adjacencyMap.get(source);
        if (neighborMap == null)
            return null;

        minEdgeValue = minimumEdgeValue();

        // Constructor de clase PathNode
        PathNode firstNode = new PathNode(source);

        PriorityQueue<PathNode> priorityQueue = new PriorityQueue<>();

        priorityQueue.add(firstNode);

        shortestCircuit = null;
        double bestCost = Double.MAX_VALUE;

        while(priorityQueue.size() > 0) {

            // Y (PathNode) = menorElemento de la cola de prioridad en funcion de
'estimatedCost'
```

```
            PathNode Y = priorityQueue.poll();

            if (Y.getEstimatedCost() >= bestCost)
                break;
            else {
                Vertex from = Y.lastVertexRes();
                // Si el numero de vertices visitados es n
                // y existe una arista que conecte 'from' con source
                if ((Y.getVisitedVertices() == numberOfVertices()) &&
                    (containsEdge(from, source))) {
                    // Actualizar 'res' en Y añadiendo el vertice 'source'
                    // Actualizar 'totalCost' en Y con Y.totalCost + weight(from,
source)

                    if (Y.getTotalCost() < bestCost) {
                        // Actualizar 'bestCost', 'shortestDistanceCircuit' y
'shortestCircuit'

                    }
                }
                else {
                    // Iterar para todos los vertices adyacentes a from,
                    // a cada vertice lo denominamos 'to'

                        if (vertice 'to' todavia no ha sido visitado en Y) { // hacer
uso de la funcion 'isVertexVisited(vertex)' de PathNode


                            PathNode X = new PathNode(Y); // Uso de constructor copia
                            // Anadir 'to' a 'res' en X
                            // Incrementar en 1 los vertices visitados en X
                            // Actualizar 'totalCost' en X con Y.totalCost +
weight(from, to)

                            // Actualizar 'estimatedCost' en X con X.totalCost +
((nVertices - X.getVisitedVertices() + 1) * minEdgeValue)


                            if (X.getEstimatedCost() < bestCost) {
                                priorityQueue.add(X);
                            }
                        }
                    }
                }
            }
        }

        return shortestCircuit;
    }
```

Otras clases necesarias

Además, necesitaréis la clase `PathNode.java`

```
protected class PathNode implements Comparable<PathNode> {
    private ArrayList<Vertex> res; // result
    private int visitedVertices; // The number of the visited vertices
    private double totalCost; // The total cost of the path taken so far
    private double estimatedCost; // Representing the lower bound of this state.
<strong>*</strong> Priority

    PathNode(Vertex vertexToVisit) {
        res = new ArrayList<Vertex>();
        res.add(vertexToVisit);
        visitedVertices = 1;
        totalCost = 0.0;
        estimatedCost = numberOfVertices() * minEdgeValue;
    }

    PathNode(PathNode parentPathNode) {
        // Constructor copia
    }

    @Override
    public int compareTo(PathNode p) {
        // El criterio de comparacion es 'estimatedCost' que se correponde con la
prioridad
    }

    public ArrayList<Vertex> getRes() {
        return res;
    }

    public void addVertexRes(Vertex v) {
        this.res.add(v);
    }

    public Vertex lastVertexRes() {
        // Devuelve el ultimo vertice que se ha anadido al camino (ultimo elemento de
'res')

    }

    public boolean isVertexVisited(Vertex v) {
        // Se ha visitado el vertice v si esta actualmente en 'res'. Una sola linea

    }

    public int getVisitedVertices() {
        return visitedVertices;
```

```
    }

    public void setVisitedVertices(int visitedVertices) {
        this.visitedVertices = visitedVertices;
    }

    public double getTotalCost() {
        return totalCost;
    }

    public void setTotalCost(double totalCost) {
        this.totalCost = totalCost;
    }

    public double getEstimatedCost() {
        return estimatedCost;
    }

    public void setEstimatedCost(double estimatedCost) {
        this.estimatedCost = estimatedCost;
    }
}
```