INTELLIGENT SYSTEMS LABORATORY PROJECT

GROUP A1-04

1.Brief explanation about the project goal

In this project we must perform a program that searchs the route between two directions of a street map.

The idea is to make this with the towns and cities of the province of Ciudad Real.

2.Tools

We choose Java as programming language to develop the project.

We have to use OSM[1] (Open Street Map) as source for the data of the cities and towns.

To manage the version control and the sharing of the project we use a repository in GitHub.

3.Implementation

OSM provides the graphs that we need in graphml files.

Graphml is a variation of the xml language used for the representation of graphs. These graphs contain nodes and edges.

The nodes are the different intersections of streets, and, the edges are the streets.

To read the data from the graph files we use a collection of interfaces and implementations built in Java called SAX (Simple API for XML)[2].

We use SAX because unlike other libraries for reading xml files, SAX processes the file by events, unlike the libraries that generate a hierarchical tree. Therefore it is ideal for manipulating large files, because it does not incur excessive memory usage. In addition, is simple to use.

We build 4 classes in Java (TSFGraph, Arc, Node and GraphHandler), according to the requirements of the problem, appart of a Principal class to launch the program.

- TSFGraph: We use this class to represent the Graph Structure and to implement the requested methods.
- Arc: This class represents the edges of the graphs (The streets), contains information about the nodes between which is, the name and the length.
- Node: This class represents the nodes of the graphs (The street junctions) and contains the coordinates of the node.
- GraphHandler: This is the class that read the graphml files using the SAX library.

1: https://www.openstreetmap.org
2: http://www.saxproject.org/

To save nodes and arcs we will make an adjacency list in which we will store the nodes with their adjacent nodes, which is the way in which the arcs are stored in an adjacency list.

So, we need a structure that allows us to introduce a node together with its adjacent ones, Dictionary and Hashmap provide us with keys and values.

We use a Hashmap to manage the “structure” of the graph, since the graph structure itself is very complex and it would be worse both in resources consumed and in the difficulty to develop the necessary methods.

We also think to use Dictionary, but it has been marked as obsolete by Java.

4.Code Highlights

```java
public boolean belongNode(String id) {
    for (Node i : nodes) {
        if (i.getID().equals(id)) {
            if (adjlist.containsKey(i)) {
                return true;
            }
        }
    }
    return false;
}
```

```java
public String[] positionNode(String id) {
    String[] pos = new String[2];
    if (!belongNode(id)) {
        return null;
    } else {
        for (Node i : nodes) {
            if (i.getID().equals(id)) {
                pos[0] = i.getX();
                pos[1] = i.getY();
            }
        }
    }
    return pos;
}
```

```java
public ArrayList<Arc> adjacentNode(String id) {
    ArrayList<Node> values = new ArrayList<Node>();
    ArrayList<Arc> adjacents = new ArrayList<Arc>();
    Node aux;
    if (!belongNode(id)) {
        return null;
    } else {
        aux = returnNode(id);
        if (adjlist.containsKey(aux)) {
            values = adjlist.get(aux);
        }
    }
    adjacents = createArcs(values, aux);
    return adjacents;
}
```

This are the 3 methods requested in the first task of the project.

And this is the constructor of the TSFGraph class, where the most important is the way we fullfill the list with the data obtained by the `readXML(filename)` method call.

```java
public TSFGraph(String filename) throws IOException, ParserConfigurationException, SAXException {
    readXML(filename);
    ArrayList<Node> aux = new ArrayList<Node>();
    for (Node i : nodes) {
        aux = new ArrayList<Node>();
        for (Arc a : arcs) {
            if (i.getID().equals(a.getSource())) {
                Node n = returnNode(a.getTarget());
                aux.add(n);
            }
        }
        adjlist.put(i, aux);
    }
}
```

5.Testing

Here we show an example of the behaviour of the program with a node of *Anchuras* extracted from the document `ExampleResults` in Campus Virtual.

```
-----   Towngraph P1 (v2.0 beta)        -----

Enter the name of the town: (or 0 to exit the program)
Anchuras
Insert the node:
4331489709
--> Belong the node to the graph? true
--> Location of the node: -4.8361395 39.4801017
--> Adjacent Nodes:
4331489709 4331489708   Calle de la Iglesia     46.118

4331489709 4331489716   Plaza España    17.119

4331489709 4331489544   Plaza España    25.208

4331489709 4331489549   Calle Amirola   137.537

Press 0 to exit the program. Press any key to try again.
```

We also test some interval of times that the program consumes, with the example above.

- Average time of reading the *Anchuras* graphml file :
  - `30292022 ns.`
- Average time of extracting the adjacent nodes:
  - `744755 ns.`

6.Participants

This project is done by:

- Sergio Herrera Piqueras
- Juan Mena Patón
- Pablo Rodríguez Solera

7.Resources

Github page:

https://github.com/A1-04/towngraph