Algorithmy Project

MICHELIN

Realised by Nicolas DEVIERS

From the eponymous well known atlas Michelin, this is a pathfinding project taking place in Santiago's metro network.
The purpose is to encounter the shortest route from two metro stations.

The algorithms used in this project are all of my own making.

In order to do this, a graph of the whole network is generated the following way : each metro line is interpreted as a doubly linked list, and then at each line intersections the nodes are merged in order to connect both lists. Therefore each node can be connected to a maximum of 4 other nodes, representing two metro lines crossing each others' path.

To start the program, the user has to give it as argument the name of the two stations he is departing from and going to. The program therefore has to be executed on a terminal or with a bash script.

Once the graph has been generated, the program will start from the destination point and spread everywhere, giving to every node a weight proportional to it's distance from the destination. So the destination's weight will be 1, and a station 5 node away will have a weight of 6 for example.
Following this logic, the weight of a node defines it's distance from the destination point, and the linked node with an inferior weight shows the way towards the destination.

For obvious reasons the executable is not in this archive. You can generate it with the Makefile, using the simple command "make a" on any linux distribution.
This project was realised with g++ 7.2.0 on Ubuntu 17.10.

The weight assignation of the nodes is handled by the following simplified algorithm , which can be found on the PathFinder.cpp file of the project :

```
Exploration (node, distance)
{
      node.weight = distance
      While node has neighbours
            If neighbour has no weight yet, or is heavier than than this node
                  Exploration(neighbour, distance + 1)

}
```

After that, the second algorithm finds back the shortest way from the start to the exit following the decreasing weights, the following way :

```
DoYouKnowDaWey (entry)
{
      For entry’s weight iterations
            While neighbor’s weight != entry’s weight -1
                  get next neighbor
            entry = next neighbor
            print entry’s new name /* it is the next node towards the exit */
}
```