METRO ATENAS

INTELIGENCIA ARTIFICIAL

UNIVERSIDAD POLITECNICA DE MADRID

Grupo 36:

- Luis Fernando Moya Lozada (coordinador, GM-II)
- Saúl Barras García
- Alejandro González Gatica
- Alberto Gómez Martinho
- Qi Ye

Índice

Introducción

El proyecto consiste en la implementación de una aplicación del metro, en concreto de Atenas, mediante una representación grafica del resultado final.

Para el desarrollo de la práctica, se ha dividido el trabajo en tres partes:

- Recolección de datos
- Algoritmo
- GUI

Recolección y esquematización de datos

Para el cálculo de los distintos trayectos se han obtenido los siguientes datos, de los que se detalla su uso y bibliografía:

- **Coordenadas** (archivo “coordenadas.csv”)
  Las coordenadas de cada estación se han obtenido manualmente mediante la búsqueda en Google Maps, y se han usado para calcular las distancias entre cada parada y las distancias óptimas para cada trayecto.
  Además, se han subdivido las coordenadas en líneas de metro para una mejor adecuación en el algoritmo:
    - MetroAzul.csv: Para la línea Azul o 1.
    - MetroRojo.csv: Para la línea Roja o 2.
    - MetroVerde.csv: Para la línea Verde o 3.

  Las versiones “graf” son para la implementación de la GUI.
- **Distancias**
  Se han calculado las distancias entre las distintas coordenadas automáticamente, mediante el uso de una función de Python “geodesic” perteneciente a la librería Geopy, la cual calcula la distancia geodésica entre dos puntos. Las distancias entre estaciones se encuentran en el archivo “distancias.csv” además de sus respectivos subconjuntos referentes a cada línea de metro:
    - DistanciaEnLineaAzul.csv.
    - DistanciaEnLineaRoja.csv.
    - DistanciaEnLineaVerde.csv.
- **Horas punta** (archivo “HorasPunta.csv”)
  Para simular un metro “real” se han recolectado datos sobre las horas punta del metro de Atenas, en los cuales se han añadido unos kilómetros extra para simular la penalización de estas horas.
  Las horas de la aplicación son simuladas y aleatorias.
- **Transbordos** (archivo “Transbordos.csv”)
  Otra de las penalizaciones son los transbordos, que en función de la distancia en la cual se encuentren las distintas líneas se van a penalizar más o menos.
  Las distancias tomadas han sido arbitrarias.

Algoritmo

Para la realización del algoritmo A*, se decidió que lo mejor era utilizar un grafo ponderado. Éste requerirá de dos clases, *Node* y *Graph*.

En cada nodo almacenaremos el nombre de la estación, el nombre de la estación anterior, es decir, el padre, y los valores ‘f’, ‘g’, y ‘h’. Además, implementamos 2 funciones de comparación, necesarias para el algoritmo, “__eq__” y “__lt__”.

Por otro lado, la clase *Graph*, representará el metro, siendo sus nodos las estaciones, y las aristas, los tramos que las conectan, con sus respectivos pesos. Por último, hemos implementado dos funciones, “connect”, la cual une dos nodos con un peso dado, y la función “get” que devuelve el peso de la arista de un nodo dado.

Gracias al trabajo hecho por parte del equipo dedicado a la recolección de datos, obtenemos de los archivos .csv los datos necesarios para formar el grafo y los extras que necesitamos, tales como distancias aéreas, coordenadas, o tiempos entre transbordos. Las distancias aéreas se almacenarán en una lista, así como las coordenadas de cada estación. A continuación, llamamos al algoritmo con la función “Algoritmo”, la cual recibirá el grafo, el nodo inicio, el nodo final, la lista con las distancias aéreas y la hora.

En primer lugar, crearemos una lista con todas las estaciones en un orden igual al que se encuentran las distancias aéreas, para posteriormente obtener su índice. La hora dada, será recibida en minutos, por lo cual, comprobaremos si está en cierto intervalo de hora para aumentar el tiempo extra del transbordo. Crearemos dos listas vacías, *open* y *closed*, además de crear los nodos inicio y final, añadiendo el inicio a *open*. Después, nos meteremos en un bucle while, que no finalizará hasta que no encuentre camino a un nodo destino. Se ordena la lista open mediante la función ‘f’, y se saca el nodo con menor valor de ‘f’ y se almacena en act_node, y si ese obtenido es igual al nodo final, se genera el path correspondiente recogiendo los nodos padres sucesivamente, y lo devuelve inverso. Si no, generamos los vecinos del nodo actual y los recorremos en un bucle for. En este, comprobaremos si dicho vecino se encuentra en la lista *closed*, de ser así, continuaremos a revisar el siguiente. Al contrario, revisaremos si entre dicho vecino y el nodo actual ha habido un transbordo, que, de ser así, almacenaremos en una variable el peso extra del susodicho. Por último, realizaremos los cálculos de ‘g’, ‘h’ y ‘f’; en ‘g’ sumaremos el ‘g’ del nodo actual, y el peso entre el nodo actual y ese vecino, más el peso del transbordo si existe. En él, almacenaremos el valor de la distancia aérea entre el nodo destino y el actual. Y en ‘f’, guardaremos la suma de ‘g’ y ‘h’; posteriormente añadiendo a *open* el vecino, si es que no está en *open* o si está pero el valor de f es menor que el de la lista.

GUI

Se ha utilizado implementación de Qt5 que ofrece Qt para generar la interfaz de usuario.
La interfaz de usuario consta de dos ventanas, una principal y otra para representar la solución a la búsqueda solicitada.

- **Ventana principal:** Donde se introducen los datos por parte del usuario para realizar la búsqueda del recorrido mínimo entre las 2 estaciones.
- **Ventana de resultado:** Donde se representa el recorrido mínimo que tiene que realizar para llegar a la estación elegida como destino desde una estación origen.

Para generar el gráfico del resultado del recorrido, se ha utilizado la librería **pandas** y **folium**. La primera que es necesaria para para generar vectores que serán usados por la segunda librería. La segunda librería se utiliza para generar un mapa sobre el que se pintarán las líneas de metro, el trayecto desde el origen al destino e información de las estaciones.

Conclusión

Con la realización del proyecto hemos concluido con las siguientes observaciones / experiencias:
El principal problema ha sido la presentación en una interfaz gráfica de la ejecución de nuestro algoritmo. Por lo cual, hemos tenido que indagar entre las distintas posibilidades gráficas de las cuales hemos elegido Qt como interfaz para realizar el proyecto, así como entre las distintas librerías gráficas de Python, como es folium, que han hecho posible el resultado final de nuestro proyecto.
La búsqueda de los datos ha presentado problemas debido a que algunos nombres no coinciden con el mapa del metro y el mapa de Google Maps de donde se han recogido los datos para posterior uso.

En definitiva la práctica supuso un desafío porque tuvimos que salir de nuestra zona de confort al utilizar un nuevo lenguaje de programación y realizar una interfaz gráfica, cosas que no se han visto durante la carrera. Pese a esto ha sido una experiencia gratificante y novedosa, que nos ha hecho aprender cosas nuevas.

Toda la bibliografía ha sido encontrada en distintas paginas de Internet de las cuales no se indican por la cantidad de éstos.