PRÁCTICA DE BÚSQUEDA
INTELIGENCIA ARTIFICIAL

José López López

Mihail Neagu

Ulises Iago Bértolo García

Marino José González García

Grupo 73

Para la elaboración de esta práctica de búsqueda hemos utilizado el lenguaje de programación Python y sus librerías *selenium*, *networkx* y *tkinter*.

Para comenzar, se han reunido todos los datos necesarios para programar el algoritmo: estaciones, sus coordenadas, líneas a las que pertenecen, todos los posibles trayectos, etc., almacenados en ficheros *.csv*. Posteriormente, para hallar todos los tiempos entre estaciones contiguas, se ha realizado un bot utilizando la librería *selenium* de Python. Este bot accede a Google Maps a través de Chrome, y utilizando las coordenadas exactas de las estaciones como argumento, obtiene el tiempo de trayecto entre ambas según los datos obtenidos a través de Maps. Después, para obtener las distancias entre las estaciones para calcular la *h(n)*, se ha utilizado un script que, utilizando unas coordenadas como argumentos, retorna la distancia entre ambas en minutos si fuéramos a la velocidad media del metro de Atenas y en línea recta, y posteriormente, las introduce en un fichero *.csv*.

Utilizando todos los datos recopilados a mano y a través del bot y el script, se desarrolla el algoritmo para obtener el trayecto óptimo entre estaciones, así como su tiempo y las líneas a través de las que se realiza. Inicialmente se cargan los datos de los ficheros *.csv* en estructuras para realizar los cálculos correspondientes mediante la función *definirestructuras()*, esta función también construye el grafo que correspondería al mapa del metro de Atenas.

Para calcular la *g(n)*, se usa una función que obtiene la distancia entre la estación de origen y la intermedia en función de la línea actual. Para ello obtiene el peso de las aristas entre ambas estaciones, y en el caso de que la línea en la que nos encontremos sea distinta a la línea entre ellas, suma al resultado el tiempo de transbordo.

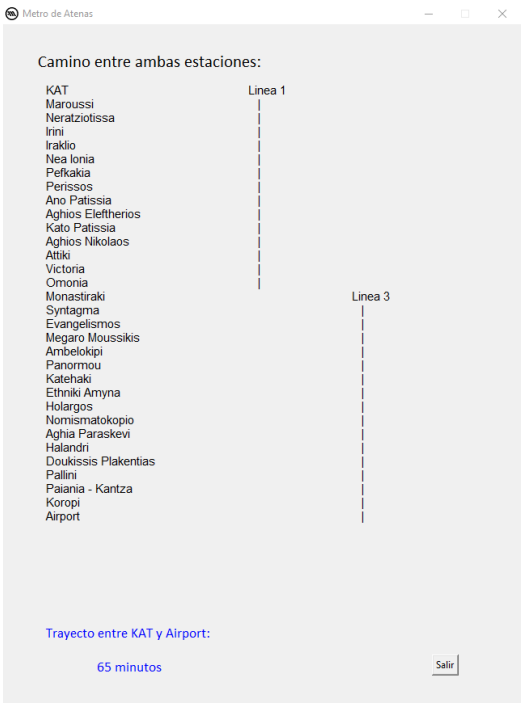
Para calcular la *h(n)*, se utiliza la función *buscarestaciones()*, a través de la cual se obtiene la distancia aproximada en minutos entre la estación intermedia y la estación destino, que se le pasan como argumentos. Además, si la línea en la que nos encontramos es distinta a las líneas entre ambas estaciones, añadimos el tiempo de transbordo al tiempo que retorna dicha función, ya que va a ser necesario realizar al menos un transbordo.

A continuación, se calcula el árbol recubridor de menor peso que contenga tanto a la estación de origen como la de destino mediante el algoritmo a*. Para ello, utilizamos un método recursivo que va obteniendo las distintas estaciones que lo componen mediante una heap queue, en la cual se introducen todos los adyacentes a la estación que se esta evaluando, cuya prioridad se obtiene calculando la función *f(n)*, que es simplemente la suma de *g(n)* y *h(n)*, y se hace después un pop para obtener la estación con menor valor de *f(n)* que será el siguiente vértice del árbol y se utilizará como argumento en la siguiente invocación del recursivo. Este método terminará una vez se añada la estación destino al árbol.

Una vez se obtiene el árbol recubridor, se obtendrá el camino que une las estaciones mediante otro método recursivo que irá sacando los adyacentes a la estación origen que se le pasa como argumento. El método comprobará que las estaciones adyacentes no estén dentro de la lista de visitados, y finalmente, en el caso de que no este, la usará como argumento en la siguiente invocación del método.

Tras obtener el camino, se podrá utiliza la función *calcularTiempo()*, que retorna el tiempo que tarda en recorrer el trayecto sumando el peso de las aristas del árbol de estaciones, y en el caso de que la línea en la que se encuentra sea distinta a la línea que hay entre las estaciones que se están evaluando, sumará también el tiempo de transbordo; así como también el método *lineasUtilizadas()* que retornara las líneas por las cuales va pasando a lo largo del trayecto.

Finalmente, tras la elaboración del algoritmo, se ha realizado una interfaz gráfica mediante la librería *tkinter* de Python, con el fin de presentar el funcionamiento del algoritmo de una manera interactiva. Esta interfaz recibe como datos las estaciones de origen y destino a través de dos campos de entrada de texto que rellenará el usuario. Si el usuario intentara introducir datos que no se encuentran dentro de la lista de estaciones, el control de errores se ocupará de indicar al usuario el uso erróneo del programa. Si se introducen datos correctos, la interfaz gráfica mostrará el trayecto seguido, así como las líneas recorridas y su tiempo de realización en una nueva ventana.



Finalmente, para el montaje del ejecutable, se ha utilizado *pyinstaller*. Se añaden todos los archivos externos para que el *pyinstaller* los monte en un fichero *.exe*, que solo es ejecutable en una maquina en la que este instalado Python.