DEPARTAMENTO DE INTELIGENCIA ARTIFICIAL (DIA)

ESCUELA TÉCNICA SUPERIOR DE INGENIEROS INFORMÁTICOS

ADRIÁN RAPOSO, ÁLVARO LÓPEZ MARTÍNEZ, CARLOS DE LEGUINA, PABLO TESORO, NICOLÁS ALCAIDE, SERGIO HERAS

PRÁCTICA INTELIGENCIA ARTIFICIAL
METRO ATENAS
Grupo 5

INTELIGENCIA ARTIFICIAL
GMI SEMESTRE 5
CURSO 2022-2023

ÍNDICE

1. Introducción 3
2. Recopilación de datos 3
3. Algoritmo A* 3
   3.1. Propiedades del Algoritmo A* 5
   3.2. Estructura de datos utilizadas en la implementación 5
4. Interfaz Gráfica 6
   4.1. Tipos De Rutas 8
5. Herramientas 9
6. Bibliografía 10

1. Introducción

El objetivo de este proyecto es diseñar una aplicación para encontrar el trayecto optimo entre dos estaciones del metro de Atenas. En nuestro proyecto, se han considerado tres parámetros distintos:

- Distancia
- Tiempo

El proyecto se ha realizado en *Python*, utilizando *Visual Studio Code*. Para el cálculo del trayecto optimo entre la estación origen y la estación destino se ha utilizado el algoritmo A*, que se expondrá más adelante.

Para mostrar la ruta mínima entre las dos estaciones se ha implementado una interfaz interactiva que como se verá más adelante permite elegir las estaciones origen y destino, así como el parámetro según el cual queremos calcular la ruta mínima. Además, contiene diferentes detalles y facilidades que hacen la interfaz más llamativa y atractiva.

2. Recopilación de datos

Primero, se realizó una búsqueda de las distancias entre las distintas estaciones con *Google Maps* y el tiempo entre estaciones se obtuvo de la página web del metro de Atenas.

A continuación, se sacaron las coordenadas geográficas de cada estación con *Google Maps* y las coordenadas de cada estación en la imagen mediante software auxiliar de *Python*.

Además, se han recopilado la línea de metro a la que corresponde cada estación.

El tratamiento de los datos ha sido realizado mediante un primer fichero metro.csv que tiene como parámetros: origen, destino, distancia y tiempo. Estos dos últimos van a ser los pesos de las aristas del grafo, será uno u otro dependiendo del parámetro escogido para hallar la ruta mínima. La estación origen y la estación destino son los nodos que unen una arista.

También se ha utilizado un fichero *.json* que se llama *coordenadas.json*, la cual almacena el nombre de la estación, latitud, longitud y la línea a la que pertenece .

Se ha recopilado también los horarios del metro Atenas en un fichero *.json* que se llama *horarios.json*.

Finalmente, la recopilación de las coordenadas de cada estación respecto de la imagen en la cual dibujaremos la ruta, en el archivo *coordImg.csv*.

Si la estación tiene posibilidad de trasbordo le hemos asignado la línea 0, que se cambiará a la hora de imprimir la ruta a la línea a la cual se hará el transbordo.

3. Algoritmo A*

El algoritmo A* es un método muy utilizado para hallar el camino óptimo entre dos puntos. Para entender dicho algoritmo cabe explicar el algoritmo A.

```
def algoritmo(self):
    sucesores = []
    solucionEncontrada = False
    self.listaAbierta.append(self.estacionActual)
    self.G.nodes[self.estacionActual]['F']=self.fheuristica(self.estacionActual)
    self.G.nodes[self.estacionActual]['G']=0
    self.G.nodes[self.estacionActual]['Padre']=None
    self.lineaAct= []
    while(solucionEncontrada == False):
        if(len(self.listaAbierta)<0):
            print("Error.")
            return
        self.estacionActual= self.valFMin()

        if self.G.nodes[self.estacionActual]['Linea'] != 0 or self.G.nodes[self.estacionActual]['Padre'] == None:
            self.lineaAct.append(self.G.nodes[self.estacionActual]['Linea'])
        elif self.G.nodes[self.estacionActual]['Linea'] == 0 and self.G.nodes[self.G.nodes[self.estacionActual]['Padre']]
            transbordosAux = self.transbordosAux()
            for y in transbordosAux:
                    if self.G.nodes[self.estacionActual]['Padre'] == y[0] and self.estacionActual == y[1]:
                        self.lineaAct.append(y[2])
        else:
            self.lineaAct.append(self.G.nodes[self.G.nodes[self.estacionActual]['Padre']]['Linea'])
```

El algoritmo A es el procedimiento general que se utiliza para la búsqueda de grafos utilizando una función para realizar la reordenación de los nodos de la lista abierta. Dicha función es la siguiente: f*(n)=g*(n)+h*(n). Donde cada función es:

- **f*(n):** es el coste real de un camino optimo desde el nodo inicial **s** a un nodo final obligando a que pase por el nodo **n.**
- Definiendo una función f que es estimador de f*: f(n)=g(n)+h(n) siendo de igual manera, g un estimador de g* y h un estimador de h*.
- **g(n):** coste del camino que va desde el nodo inicial **s** al nodo **n** en el árbol de búsqueda que es el camino de coste más bajo encontrado en este momento.
- **h(n):** función heurística, siendo concretamente: la acumulación de g(n)+distancia euclídea hasta el nodo de destino. El cálculo de esta funcion depende de si el usuario desea calcular la ruta en tiempo o en distancia.

A continuación, el algoritmo A selecciona para ser expandido el nodo en abierta que tiene un valor más pequeño en la función f. En el caso de que h(n) es menor o igual que h*(n) para todo **n** entonces el algoritmo se llama A*.

Para hallar en el camino óptimo en este proyecto se crea una lista abierta y una lista cerrada. En la lista abierta se añade la estación en la que se encuentra, mientras que en la lista cerrada se mete la estación que tenga menor valor de f(n), de entre todas las estaciones que se puede llegar desde la actual (es decir, nodos hijos). Cada vez que se añade un nodo en la lista cerrada se elimina de la lista abierta. Se realizará esto hasta que la estación en la que nos encontramos sea la estación destino.

Esto se hace en bucle, en el que en cada iteración se comprueba si hemos llegado a la estación origen. En caso contrario, nos sacamos las estaciones contiguas con la actual ("estaciones hijas").

Para cada nueva estación comprobamos que no esté en la lista cerrada, si lo estuviese no haríamos nada con esa estación. Sin embargo, si no está en la lista cerrada comprobamos si está en la lista abierta. Si lo está, le asignamos el valor de 'f' óptimo. Si no estuviese en la lista abierta la añadimos con su valor de 'f' correspondiente. Una vez hecho esto en bucle llegaremos hasta la estación destino.

El cálculo de f(n), como hemos mencionado antes, es la suma de g(n) + h(n).

Para el cálculo de h(n) nos hemos creado una función que calcula la función heurística. En ella tenemos en cuenta la latitud y longitud de la estación en la que nos encontramos en ese momento y de la estación destino. Además, debemos tener saber el radio de la tierra y las diferencias de latitud y longitud entre ambas estaciones. Con estos datos ya podemos calcular 'h' mediante la fórmula *Haversine*. Si el criterio elegido es el tiempo, en vez de devolver la distancia entre los dos puntos, se devolverá el tiempo estimado que tardaría el tren en recorrer esta distancia, suponiendo una velocidad del tren de 1.3 kilómetros por minuto.

El cálculo de 'g' de una estación concreta será la suma de la 'g' de la estación "padre" y del valor de la arista en la estación actual. El valor de esta arista será el tiempo o la distancia, dependiendo del parámetro elegido, que hay hasta esta estación. Si el criterio elegido es el tiempo, y el trayecto a dicha estación requiere de un trasbordo, se añadirá una penalización puesto que para el trasbordo siempre supondrá más tiempo que no cambiar de línea.

Finalmente, para saber cuál es el recorrido mínimo vamos a recorrer la lista cerrada empezando por la última posición. Primero vemos si la estación destino tiene un "padre", si lo tiene nos situaremos en esa estación y haremos lo mismo (comprobar si tiene un "padre"). Haremos esto hasta llegar a una estación que no tenga "padre", esta será la estación origen. Todas estas estaciones que hemos ido comprobando si tienen "padre" las vamos añadiendo en una lista. Cuando terminemos las estaciones que estén en dicha lista serán las estaciones del recorrido óptimo.

Si el criterio elegido es el tiempo, entonces después de calcular el recorrido se procederá a calcular los tiempos de espera en la estación origen y en los trasbordo, en el caso de que haya, añadiéndoselo al tiempo de trayecto obtenido.

3.1. Propiedades del Algoritmo A*

- Completitud: es un algoritmo que termina con una solución si esta existe
- Admisibilidad: es admisible ya que asegura encontrar una solución óptima si esta existe
- Eficiencia: Un algoritmo A1 es más eficiente que otro A2 si A1 está más informado que A2, así cada nodo desarrollado por A1 también es desarrollado por A2. A2 desarrolla, al menos tantos nodos como A1
- Optimalidad: es óptimo ya que, dentro de un conjunto de procedimientos, es más eficiente que todos los elementos de dicho conjunto

3.2. Estructura de datos utilizadas en la implementación

- Archivos *.csv*:

```
ORIGEN;DESTINO;DISTANCIA;TIEMPO
Piraeus;Faliro;2.2;4
Faliro;Moschato;1.90;3
Moschato;Kallithea;1.65;2
Kallithea;Tavros;0.65;2
Tavros;Petralona;0.81;2
Petralona;Thissio;1.55;2
Thissio;Monastiraki;0.46;2
Monastiraki;Omonia;0.93;2
Omonia;Victoria;1.00;2
Victoria;Attiki;1.96;2
Attiki;Aghios Nikolaos;0.93;2
Aghios Nikolaos;Kato Patissia;0.55;2
Kato Patissia;Aghios Eleftherios;1.01;2
Aghios Eleftherios;Ano Patissia;0.55;1
Ano Patissia;Perissos;1.28;3
Perissos;Pefkakia;0.69;1
Pefkakia;Nea Ionia;0.70;2
```

- Archivos *.json*

```
[
    {
        "Station": "Piraeus",
        "Latitude": 37.948116,
        "Longitude": 23.643272,
        "Line": 1
    },
    {
        "Station": "Faliro",
        "Latitude": 37.944988,
        "Longitude": 23.665225,
        "Line": 1
    },
    {
        "Station": "Moschato",
        "Latitude": 37.955060,
        "Longitude": 23.679700,
        "Line": 1
    },
```

- Listas para almacenar los archivos *.csv* y *.json*.
- Listas de listas en *Python*
- *Data-frame*
- *Datetime* (manipulación de variables temporales)

4. Interfaz Gráfica

La interfaz gráfica de cualquier programa es fundamental para la facilidad del usuario. Se ha intentado que sea lo más intuitiva posible dándole especial importancia a las imágenes, colores, recorridos interactivos, … Se ha querido hacer un balance entre lo atractivo y lo funcional. El programa aparecería tal que así:

La pestaña principal se compone de dos partes. En la parte de la izquierda vienen las opciones que tiene el usuario para modelizar su ruta abriendo las pestañas de selección de origen y destino. También puede elegir el parámetro para priorizar su ruta, tiempo o distancia, lo cual variará la ruta, así como un botón que indicará los horarios de la salida de los trenes de cada línea, para cada día de la semana. Es importante elegir una opción de cada campo ya que, si no se selecciona ninguna, saldrá un error que se explicará más adelante.

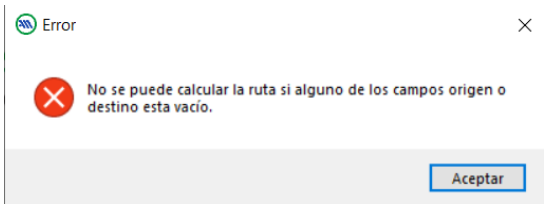
La parte de la derecha corresponde con el plano de metro de Atenas para que el usuario pueda escoger de manera interactiva su ruta. Se ha conseguido implementar un formato en el que el usuario puede clicar en las estaciones que quiera.

Una vez que el usuario selecciona la estación origen, aparecerá una imagen de un tren en dicha estación indicando que es la estación origen. Dicha imagen es la siguiente:

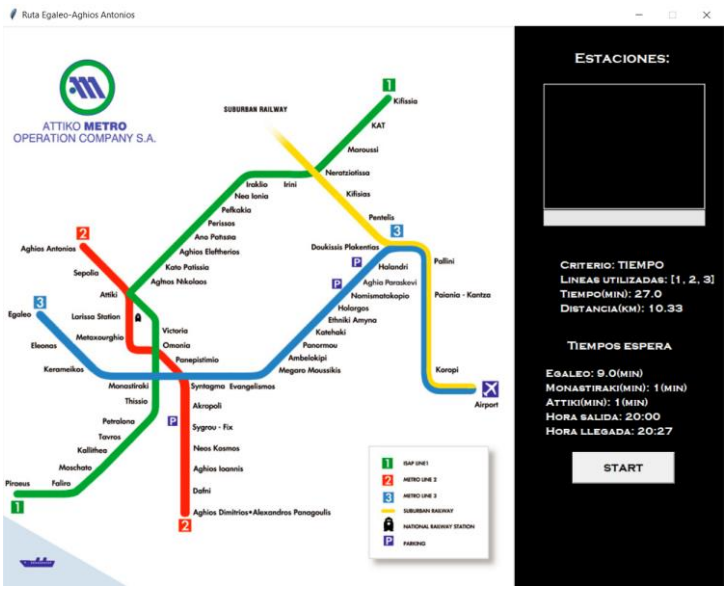
Asimismo, cuando el usuario selección la estación destino, aparecerá en esta una imagen de una bandera de cuadros blancos y negros indicando que es la estación destino. Dicha imagen es la siguiente:

Un ejemplo de cómo quedaría después de seleccionar la estación origen (*Sygrou - Fix*) y destino (*Iraklio*) sería el siguiente:

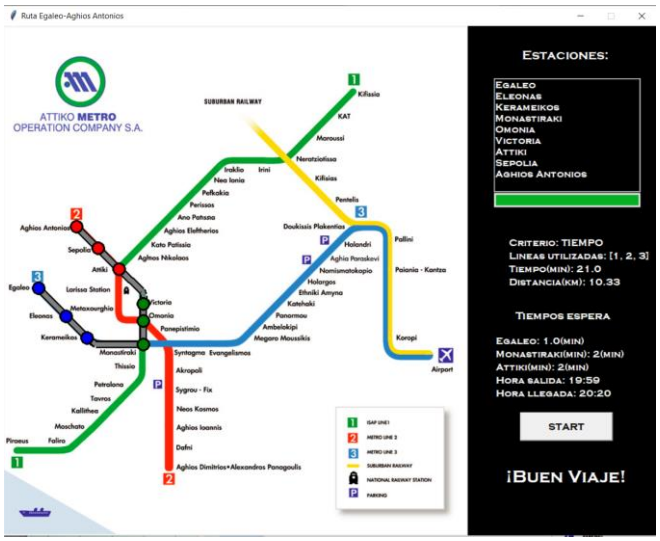
Cuando no se elige algún campo en la pestaña principal, necesario para la implementación de la ruta, saldrá un aviso de error indicando de que la ruta no se pudo crear debido a dichas razones. El aviso es el siguiente:



Una vez elegida la ruta correctamente, se nos abrirá una ventana emergente. A la derecha nos indica el criterio que ha elegido el usuario, el tiempo que va a tardar, incluyendo el tiempo de espera que supondrá coger el tren dependiendo de la hora actual, la distancia entre la estación origen y destino, y las líneas utilizadas. Justo debajo hay un botón que pone *START* que, cuando es pulsado, se hará un recorrido interactivo en el plano y en el recuadro de estaciones saldrá un listado de las estaciones que se recorren hasta llegar al destino, así como una barra de progreso. A continuación, se muestra un caso real haciendo el proceso de cuando se abren las nuevas pestañas:



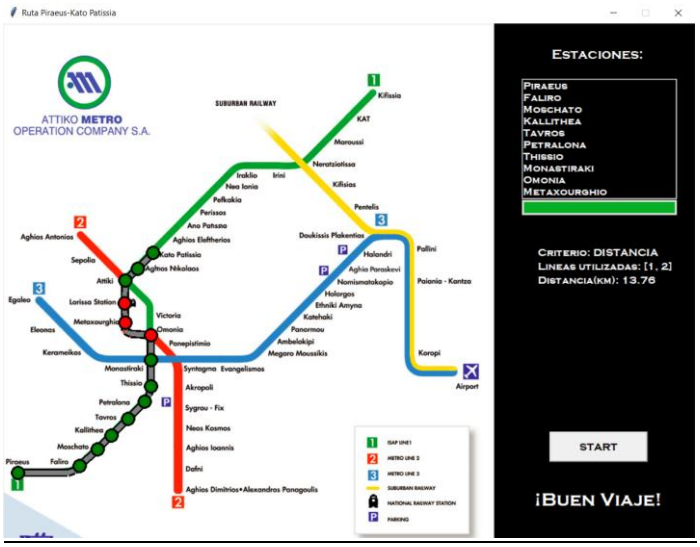
Pulsando el botón START:



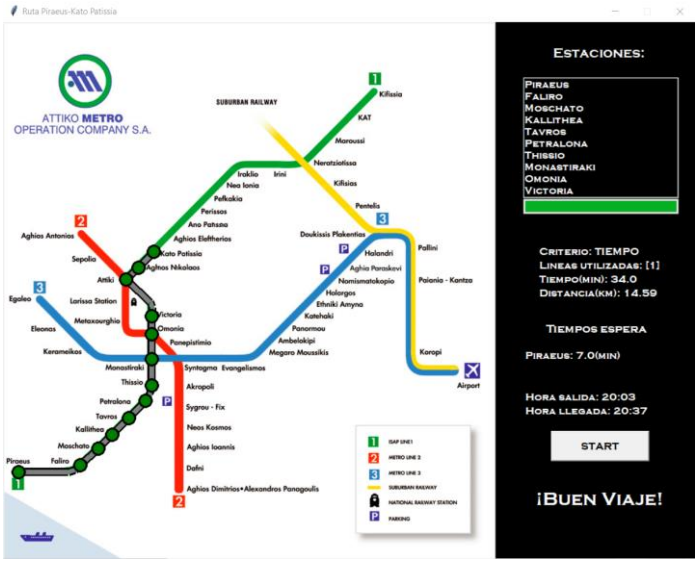
4.1. Tipos De Rutas

Dependiendo del criterio y la conveniencia del usuario, se puede escoger dependiendo de si da prioridad a la distancia o al tiempo.

Ruta Siguiendo El Criterio Distancia:



Ruta Siguiendo El Criterio Tiempo:



Como se puede observar, dependiendo de las preferencias del usuario, la ruta va a ir variando. Se ha escogido este ejemplo ya que se puede ver claramente que la ruta mínima varía dependiendo del parámetro escogido (tiempo o distancia).

5. Herramientas

En las directrices de la práctica no se especificaba ningún entorno de desarrollo o herramienta grafica para su implementación. Por lo tanto, se ha decidido hacer uso de *Python* ya que es asequible de usar y contiene una gran cantidad de herramientas útiles para el proyecto. Se han importado diversas librerías del propio *Python* como *networkx*, *tkinter*, *pandas*, *datetime*, entre otras, para que sea más versátil hacer la práctica.

Cabe mencionar que hemos valorado otras opciones como *Java*, descartadas por las facilidades que *Python* ofrecía. Se ha utilizado el editor de código *Visual Studio Code* y para realizar el trabajo de forma paralela se ha ido subiendo el trabajo a *GitHub* de forma que cada integrante del grupo pudiera ir trabajando por su parte.

La librería *networkx* se ha implementado para poder utilizar grafos en el programa y también utilizar los métodos que ofrece para obtener información del grafo.

La interfaz se ha realizado con la librería *tkinter* que permitirá hacer una interfaz interactiva y llamativa. Se ha dado especial uso a los *frames* y a los botones, que son herramientas de la propia librería. Los *frames* son esenciales, permitían mostrar imágenes, como la del propio mapa de Atenas, y texto para aclarar lo que el usuario está viendo. Por otro lado, los botones permiten seleccionar cualquier estación del mapa. También se utilizan para que aparezca una ventana emergente con la ruta óptima obtenida. Además de los *frames* y de los botones se han utilizado más funciones de *tkinter* que perfeccionan la interfaz.

También se han usado otras herramientas de *Google* como *Google Maps* para poder familiarizarse con las redes de transporte, los tiempos y distancias entre las distintas estaciones para ver cuál podía ser la mejor manera de implementación.

Se ha recopilado información de la página web del Metro de Atenas para estudiar las distancias, los tiempos entre estaciones, …

6. Bibliografía

Para hallar las coordenadas geográficas de las estaciones del metro de Atenas y las distancia entre ellas, se ha utilizado Google Maps (https://www.google.es/maps/?hl=es).

Para la realización de la interfaz, ha servido de ayuda el canal de youtube p*ildorasinformaticas*.

Link al canal: https://www.youtube.com/@pildorasinformaticas

Para saber el tiempo que se tarda en llegar de una estación a otra, se ha usado la página web del metro de Atenas: https://stasy.gr/en/timetables

Para calcular la distancia entre dos coordenadas geográficas, hicimos uso de la Fórmula de Haversine, que encontramos en esta página: https://www.genbeta.com/desarrollo/como-calcular-la-distancia-entre-dos-puntos-geograficos-en-c-formula-de-haversine