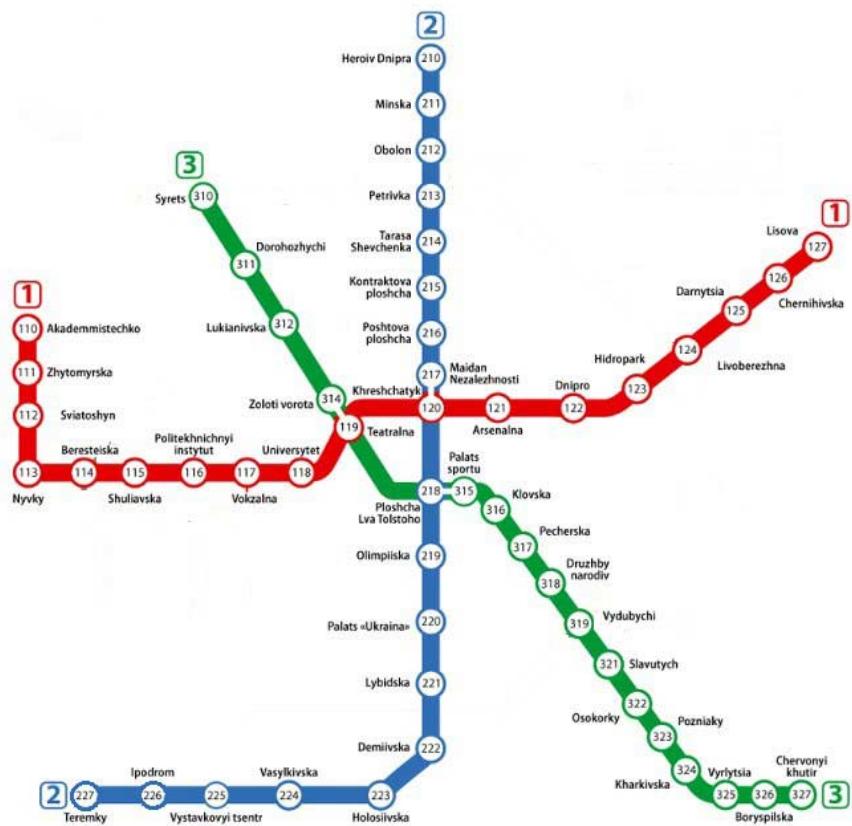


# Memoria Trabajo Inteligencia Artificial

GMI 2021-2022

Grupo 2



# Índice:

<b>Introducción</b>	<b>4</b>
<b>Obtención de datos</b>	<b>4</b>
1.- Tiempos estimados	4
2.- Horario de apertura	5
3.- Distancias aéreas	5
Fórmula de Haversine/Semiverseno	6
<b>Desarrollo del algoritmo A*</b>	<b>7</b>
1.- Tipos de datos	7
2.- Algoritmo	7
<b>Herramientas utilizadas</b>	<b>8</b>
1.- Python	8
2.- Módulos	8
tkinter	8
datetime	8
math	8
ttkthemes	8
canvas	9
<b>Interfaz gráfica</b>	<b>10</b>
1.- Selección de las paradas	10
2.- Hora:	10
4.- Mapa del metro	11
5.- Visualización de la ruta	11
5.1.- En la propia imagen.	11
5.2.- Por escrito.	11
6.- Avisos de error	12
7.- Menú	12
7.1.- Nuevo y cerrar	12
7.2.- Apariencia	12
7.3.- Información: acerca de e información	13
Visualización de las interfaces en diferentes Sistemas Operativos	14
1.- Windows:	14
2.- Mac:	14
3.- Linux:	15
<b>Problemas y mejoras</b>	<b>15</b>
1.- Problemas surgidos	15
2.- Mejoras	16

# Introducción

El problema inicial del grupo fue plantear el problema. Al contrario de los ejercicios propuestos en clase, en los que empleábamos el A\* para obtener el camino más corto entre dos nodos, en este ejercicio, se nos pide encontrar la ruta óptima entre paradas de metro.

No tiene sentido obtener la distancia más corta, ya que lo que queremos es llegar cuanto antes al destino. Luego los valores que se les dan a las conexiones entre nodos no son las distancias, sino los tiempos. Es decir, el valor entre dos nodos adyacentes es el tiempo que tardas en ir de uno al otro. De esta forma, la ruta óptima es la ruta de tiempo mínimo.

Luego, el primer paso del proyecto fue obtener estos datos.

## Obtención de datos

### 1.- Tiempos estimados

Para calcular el tiempo real que tardamos en desplazarnos de una parada a otra, utilizamos Easyway<sup>1</sup>. Por cada parada en nuestro mapa, introducimos sus adyacentes en la página y obtuvimos los tiempos, los cuales empleamos para crear la tabla G.

```
self.tablaG = {
    '110': [(('111', 3)],
    '111': [(('110', 3), ('112', 3)],
    '112': [(('111', 3), ('113', 2))],
    '113': [(('112', 2), ('114', 2))],
    '114': [(('113', 2), ('115', 4))],
    '115': [(('114', 4), ('116', 2))],
    '116': [(('115', 2), ('117', 4))],
    '117': [(('116', 4), ('118', 2))],
    '118': [(('117', 2), ('119', 1))],
    '119': [(('118', 1), ('120', 1), ('314', transbordos(hora, dia, '314')))],
    '120': [(('119', 1), ('121', 3), ('217', transbordos(hora, dia, '217')))],
    '121': [(('120', 3), ('122', 2))],
    '122': [(('121', 2), ('123', 2))],
    '123': [(('122', 2), ('124', 3))],
    '124': [(('123', 3), ('125', 2))],
    '125': [(('124', 2), ('126', 2))],
    '126': [(('125', 2), ('127', 2))],
    '127': [(('126', 2))],
```

```
def transbordos(hora, dia, parada):
    if(dia == 'SABADO' or dia == 'DOMINGO'):
        if(parada == '119'):
            res = 3.5
        elif(parada == '314'):
            res = 6
        elif(parada == '120'):
            res = 3.5
        elif(parada == '217'):
            res = 4.5
        elif(parada == '218'):
            res = 4.5
        else:
            res = 6
    else:
        if((hora >= 7 and hora < 10) or (hora >= 17 and hora < 20)):
            if(parada == '119'):
                res = 2
            elif(parada == '314'):
                res = 2.5
            elif(parada == '120'):
                res = 2
            elif(parada == '217'):
                res = 2.25
            elif(parada == '218'):
                res = 2.25
            else:
                res = 2.5
        else:
            if(parada == '119'):
                res = 3.5
            elif(parada == '314'):
                res = 5
            elif(parada == '120'):
                res = 3.5
            elif(parada == '217'):
                res = 5
            elif(parada == '218'):
                res = 5
            else:
                res = 5
    return res + 1
```

En las paradas “especiales”, que son transbordos, el tiempo introducido lo calculamos sumando el tiempo andando entre las estaciones, y el tiempo estimado hasta que llegue el metro a la nueva parada.

Los datos de tiempo andando se han obtenido mediante EasyWay, y la frecuencia mediante datos de Wikipedia<sup>2</sup>. Estos datos dependen de variables como la hora, el día de la semana y la línea de metro.

<sup>1</sup> <https://www.eway.in.ua/en/cities/kyiv>

<sup>2</sup> [https://es.wikipedia.org/wiki/Metro\\_de\\_Kiev](https://es.wikipedia.org/wiki/Metro_de_Kiev)

Línea	Frecuencia	
Línea 1	Hora punta	2 mins
	Normalidad	3:30 mins
	Fines de semana	3:30 mins
Línea 2	Hora punta	2:15 mins
	Normalidad	5 mins
	Fines de semana	4:30 mins
Línea 3	Hora punta	2:30 mins
	Normalidad	5 mins
	Fines de semana	6 mins

## 2.- Horario de apertura

Línea 1: 5:27-00:18

Línea 2: 5:22-00:16

Línea 3: 5:20-00:17

Datos recogidos de la guía de metro de Kiev<sup>3</sup>. Con esta información indicamos al usuario si el metro está cerrado a la hora que quiere buscar la ruta óptima, o si el metro va a cerrar antes de que llegue al destino.

## 3.- Distancias aéreas

Para el cálculo del tiempo aéreo, decidimos utilizar la siguiente fórmula:

Siendo:

- V la velocidad media del metro
- D la distancia aérea real
- T el tiempo aéreo

$$V = \frac{D}{T}$$

De este modo obtenemos un “tiempo aéreo”, que empleamos para calcular la tabla H.

Para calcular la distancia aérea, en un principio pensamos en obtener las distancias usando la opción de Google Maps. Esto consistía en seleccionar todas las combinaciones de paradas, de dos en dos, y seleccionar la opción de calcular distancia. Sin embargo, pronto nos dimos cuenta de que no era factible, ya que tendríamos que realizar el proceso  $C(52,2) = 1326$  veces. Por tanto, optamos por utilizar la fórmula de Haversine.

- Fórmula de Haversine/Semiverseno

```
# Línea 1
p110 = Punto(50.46494012642525, 30.355093416884497, 23, 253)
p111 = Punto(50.456247626897266, 30.365447716885082, 23, 286)
p112 = Punto(50.457766184988856, 30.390651834628162, 23, 318)
p113 = Punto(50.45864903879728, 30.40437299905491, 23, 362)
p114 = Punto(50.45931819257611, 30.41863764016107, 65, 362)
p115 = Punto(50.45422742536485, 30.44862354015915, 104, 362)
p116 = Punto(50.45112945229003, 30.46429282850334, 148, 362)
p117 = Punto(50.44133462518602, 30.48996366960698, 188, 361)
p118 = Punto(50.44326719817879, 30.50463264015914, 229, 361)
p119 = Punto(50.44479360629467, 30.51552534016359, 265, 327)
p120 = Punto(50.44734465226959, 30.52275835795556, 327, 312)
p121 = Punto(50.44306386039469, 30.54715874016503, 378, 312)
p122 = Punto(50.441227639163415, 30.559338500883747, 434, 312)
p123 = Punto(50.445977325247455, 30.576883999054036, 482, 298)
p124 = Punto(50.45186375229452, 30.598156428504502, 520, 268)
p125 = Punto(50.45595732538683, 30.61296942850532, 557, 239)
p126 = Punto(50.459892606100276, 30.630291328508495, 588, 214)
p127 = Punto(50.46465869866755, 30.645565028501128, 617, 192)
```

<https://www.kievkyivukraine.com/es/metro-kiev.cs.html>

La fórmula de Haversine permite hallar la distancia entre dos puntos dadas las coordenadas de latitud y longitud, que conseguimos utilizando Google Maps. Así, reducimos de tener que hacer 1326 pasos para obtener las distancias aéreas, a 52 pasos más aplicar la fórmula:

$$d = 2r \arcsin \left( \sqrt{\sin^2 \left( \frac{\phi_2 - \phi_1}{2} \right) + \cos(\phi_1) \cos(\phi_2) \sin^2 \left( \frac{\lambda_2 - \lambda_1}{2} \right)} \right)$$

Siendo:

- d la distancia que queremos hallar
- r el radio de la Tierra (6372.795 km)
- $(\phi_1, \lambda_1)$  las coordenadas de latitud y altitud del primer punto
- $(\phi_2, \lambda_2)$  las coordenadas de latitud y altitud del segundo punto

```
def distanciaH(lat1, lon1, lat2, lon2):
    rad = mt.pi/180
    dlat = lat2-lat1
    dlon = lon2-lon1
    R = 6372.795477598
    a = (mt.sin(rad*dlat/2))**2 + mt.cos(rad*lat1) * \
        mt.cos(rad*lat2)*(mt.sin(rad*dlon/2))**2
    distancia = 2*R*mt.asin(mt.sqrt(a))
    return (distancia/37.5)*60
```

```
self.tablaH = {
    '110': distanciaH(parada.getX(), parada.getY(), p110.getX(), p110.getY()),
    '111': distanciaH(parada.getX(), parada.getY(), p111.getX(), p111.getY()),
    '112': distanciaH(parada.getX(), parada.getY(), p112.getX(), p112.getY()),
    '113': distanciaH(parada.getX(), parada.getY(), p113.getX(), p113.getY()),
    '114': distanciaH(parada.getX(), parada.getY(), p114.getX(), p114.getY()),
    '115': distanciaH(parada.getX(), parada.getY(), p115.getX(), p115.getY()),
    '116': distanciaH(parada.getX(), parada.getY(), p116.getX(), p116.getY()),
    '117': distanciaH(parada.getX(), parada.getY(), p117.getX(), p117.getY()),
    '118': distanciaH(parada.getX(), parada.getY(), p118.getX(), p118.getY()),
    '119': distanciaH(parada.getX(), parada.getY(), p119.getX(), p119.getY()),
    '120': distanciaH(parada.getX(), parada.getY(), p120.getX(), p120.getY()),
    '121': distanciaH(parada.getX(), parada.getY(), p121.getX(), p121.getY()),
    '122': distanciaH(parada.getX(), parada.getY(), p122.getX(), p122.getY()),
    '123': distanciaH(parada.getX(), parada.getY(), p123.getX(), p123.getY()),
    '124': distanciaH(parada.getX(), parada.getY(), p124.getX(), p124.getY()),
    '125': distanciaH(parada.getX(), parada.getY(), p125.getX(), p125.getY()),
    '126': distanciaH(parada.getX(), parada.getY(), p126.getX(), p126.getY()),
    '127': distanciaH(parada.getX(), parada.getY(), p127.getX(), p127.getY()),
```

Una vez obtenida la distancia aérea, se divide entre  $37.5 \text{ kmh}^{-1}$ , que es la velocidad media del metro. De esta forma se obtiene el valor entre dos paradas. Usando el nodo de origen, se completa la tabla H con las distancias entre ese nodo y el resto

## Desarrollo del algoritmo A\*

### 1.- Tipos de datos

Para el desarrollo del algoritmo A\* hemos implementado en nuestro programa dos tablas G y H. Estas tablas se traducen al código mediante diccionarios. Cada nodo de nuestro problema se corresponde con una *key* en el diccionario. En la tabla H, asociada a cada *key* se encuentra la distancia aérea entre ese nodo y el nodo destino. En la tabla G cada nodo corresponde con una *key* igualmente, pero tiene asociada tantos pares (clave, valor) como nodos adyacentes tenga, siendo “clave” la *key* del nodo adyacente y “valor”, el valor del tiempo real que tardas en llegar desde nuestro nodo al nodo “clave”.

Por otro lado, hemos añadido un diccionario para guardar las paradas en estado abierto, donde cada *key* corresponde con un nodo, y su valor es el valor actual de F en ese nodo. Para el conjunto de nodos cerrados, hemos implementado un set (conjunto de elementos).

Por último, para determinar el camino “solución”, hemos implementado otro diccionario que simula un árbol de nodos. Cada *key* corresponde con un nodo y su valor equivale a su nodo “padre”.

## 2.- Algoritmo

Mediante dichas estructuras de datos, la implementación del algoritmo no tiene ninguna dificultad. Siguiendo los pasos aprendidos en las clases teóricas, en cada iteración del algoritmo:

- Se obtiene el nodo de la lista de abiertos con menor valor de F mediante la iteración de las *keys* del diccionario.
- Se analizan todos los hijos de ese nodo: Se itera sobre cada hijo, obtenido de la tabla G, se comprueba que ese hijo no pertenece al conjunto de nodos cerrados, y en caso de que no pertenezca se calcula su nuevo valor de G. Si dicho hijo pertenece ya al diccionario de abiertos, se compara su nuevo valor y su antiguo, y nos quedamos con el menor. En caso de no pertenecer, se añade directamente a la lista de abiertos. En los dos casos, ese nodo hijo se añade como key en el diccionario que simula el árbol con nuestro nodo padre como valor. En caso de que el nodo ya estuviera en el conjunto de abiertos, si cambiamos su valor de G también cambia su padre, por lo que tenemos que actualizar su valor en el árbol.
- Por último, se añade nuestro nodo al conjunto de cerrados y se elimina del diccionario de abiertos.

Una vez finalizado el algoritmo, queda como solución un árbol. Mediante iteraciones desde el nodo destino obteniendo el padre de cada nodo en cada iteración podemos señalar el camino que se ha seguido para encontrar la ruta óptima.

## Herramientas utilizadas

### 1.- Python

Decidimos utilizar Python por recomendación del profesor, ya que se nos dijo que tiene una gran cantidad de librerías y de facilidades para hacer la interfaz gráfica.

El problema con esto es que nunca habíamos programado (más allá de realizar algún laboratorio) en Python. Esto supuso que tuviésemos que buscarnos la vida para aprender este lenguaje y programar nuestro trabajo. Aún así, se nos ha hecho bastante intuitivo, y hemos aprendido varias de las diferencias entre lenguajes a base de fallos y errores.

### 2.- Módulos

- **tkinter**

Decidimos utilizar Tkinter porque buscando cómo implementar una interfaz gráfica con Python, era una de las librerías más recomendadas por la gente; se decía que era intuitiva y bastante fácil de usar.

A parte de lo básico como crear ventanas, cambiar el icono de la ventana, añadir imágenes, agregar un menú etc., utilizamos varios widgets: Button, Label, Spinbox y Entrybox, los cuales sacaban datos por pantalla, almacenaban datos o manipulaban funciones. Con todo esto, crear un frame y una estructura del programa que es intuitiva para el usuario.

Cabe destacar que hemos utilizado estos widgets, concretamente del módulo tkinter.ttk. A pesar de que el propio tkinter tiene estos widgets, hemos decidido usarlos del ttk debido a que tiene los estilos predefinidos. Por lo que no nos hemos tenido que preocupar demasiado por su apariencia.

- **datetime**

Utilizado para obtener la hora real y ponerla como la hora por defecto al abrir nuestro programa, o al darle la opción de “Nuevo”, que actualiza la hora y el día.

- **math**

Empleado para las funciones trigonométricas y constantes matemáticas como  $\pi$ , para el cálculo de distancias.

- **ttkthemes**

Este es un módulo que no viene instalado cuando se descarga Python. Aunque está sobre todo integrado para Windows, la mayoría de las funciones (no todas) también están para Mac y Linux. Lo hemos utilizado para obtener diferentes apariencias, como modo Breeze (tono azul), Equilux (modo oscuro) o Yaru (inspirada en la temática de Linux), entre otras. Nos ha permitido conseguir una interfaz gráfica bastante moderna y bonita en comparación con la predefinida.

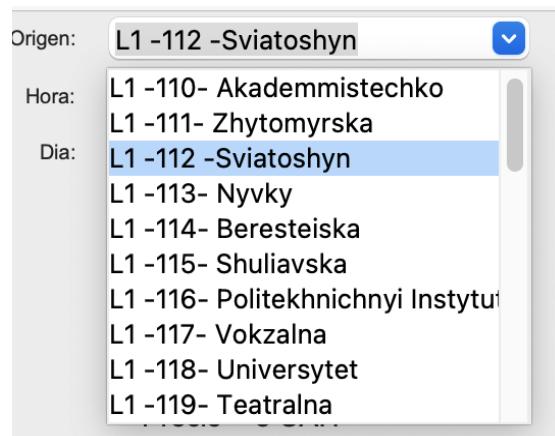
- **canvas**

Introducido para insertar imágenes, como la del mapa del metro de Kiev, y para representar la ruta solución. Para ello, hemos obtenido las coordenadas de las paradas respecto de la imagen y hemos pintado las flechas, dando el origen y el final, utilizando splines cuadráticos para que las flechas no siempre fueran rectas; si no que se adaptaran a la forma de las líneas de metro.

## Interfaz gráfica

### 1.- Selección de las paradas

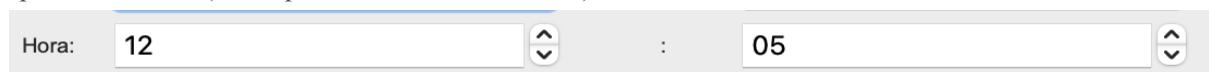
Utilizamos Comboboxes para las paradas. Esta decisión fue debido a que no era factible pedirle al usuario que introdujera el nombre entero de las paradas. Tampoco era muy cómodo introducir el código de parada, ya que eso implicaría pedir al usuario ver qué código tiene cada parada. Por tanto, al final optamos por el Combobox, con el que sale un menú desplegable y deslizable. Esto a su vez, reduce la posibilidad de que el usuario introduzca un error, por ejemplo, que se introdujera un código no existente.



### 2.- Hora:

Empleamos un Spinbox para la hora y el minuto (parámetros de los que depende el tiempo aproximado en llegar). La ventaja de este widget es que no necesitan utilizar el teclado, pueden utilizar las flechas de al lado para incrementar o decrementar el valor, pero también se puede escribir por teclado. Cabe destacar que si por ejemplo tenemos la hora 23 e incrementamos el valor se convierte en 0.

Por defecto, la hora al abrir el programa es la hora actual. Asimismo, al reiniciar el programa con la opción “Nuevo” (de la que hablaremos más tarde) se actualiza la hora.



### 3.- Día:

Utilizado un OptionMenu. La diferencia entre el OptionMenu y el Combobox es que este no es deslizable, ya que son solo 7 opciones.



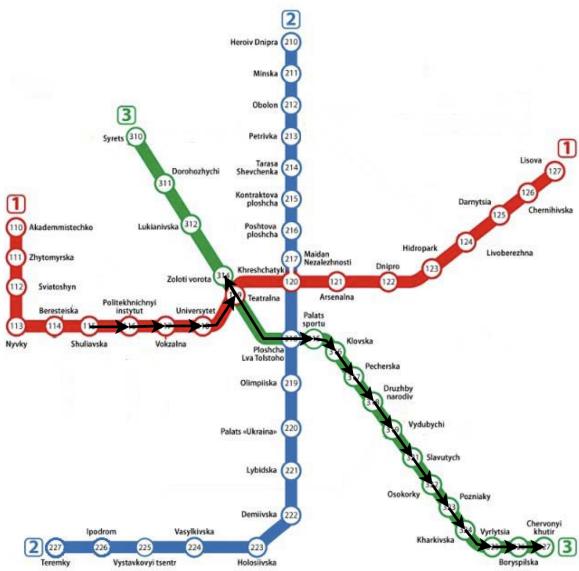
#### **4.- Mapa del metro**

Se muestra la imagen del mapa del metro para que el usuario lo tenga a mano siempre que lo necesite.

## 5.- Visualización de la ruta

### *5.1.- En la propia imagen.*

Para visualizar la solución, se muestra el itinerario con flechas en negro encima del mapa uniendo las paradas.



Para realizar esto, obtuvimos las coordenadas de las paradas respecto de la imagen utilizando un programa de visualización de imágenes Gimp.

### **5.2.- Por escrito.**

Precio = 8 UAH

La ruta óptima es:

- L1 -111- Zhytomyrska
- L1 -112 -Sviatoshyn
- L1 -113- Nyvky
- L1 -114- Beresteska
- L1 -115- Shuliatyska
- L1 -116- Politehnichnyi Instytut
- L1 -117- Vokzalna
- L1 -118- Universitet
- L1 -119- Teatralna
- L3 -314- Zoloti Vorota
- L3 -312- Lukianivska
- L3 -311- Dorogozhychi
- L3 -310- Svyrets

El tiempo total estimado es de 39 mins  
1 mins andando, 33.0 mins en metro  
Llegará aproximadamente a las 20:55

Transbordos: sí  
Tiempo estimado de 5 mins

Mostramos las paradas por las que pasa en orden. Además, indicamos el coste<sup>4</sup> de la ruta, la duración del trayecto, el tiempo de llegada estimado, tiempo andando, tiempo en el metro, si hay que hacer transbordo y su duración.

<sup>4</sup> [https://es.wikipedia.org/wiki/Metro\\_de\\_Kiev](https://es.wikipedia.org/wiki/Metro_de_Kiev)

## 6.- Avisos de error

A continuación indicamos los errores que puede realizar el usuario.

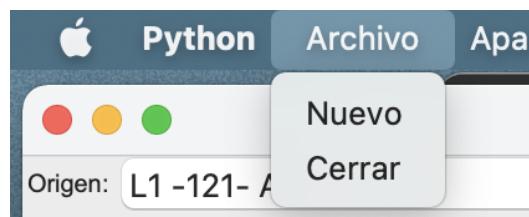
- *No se ha introducido la parada origen o destino:* a pesar de haber utilizado un Combobox con el que salen las opciones desplegables, no hay paradas por defecto, por lo que si no se introduce alguna parada saldría error.
- *Hora no válida:* ocurre si al introducir una hora superior a 23 o minuto superior a 59. Solo ocurre este error introduciéndolo por teclado, ya que las flechas del Combobox ya se encargan de que no se pasen del intervalo.
- *El metro no abre a la hora seleccionada:* con los datos de horario de apertura que mostramos en apartados anteriores, indicamos al usuario de si está el metro abierto a la hora que consulta la ruta. A pesar de mostrar el error, mostramos cuál sería la ruta, suponiendo que se realiza el trayecto a la primera hora de apertura.
- *El metro cerrará antes de que se llegue al destino:* el metro está abierto a la hora seleccionada, pero cierra antes de llegar al destino.

## 7.- Menú

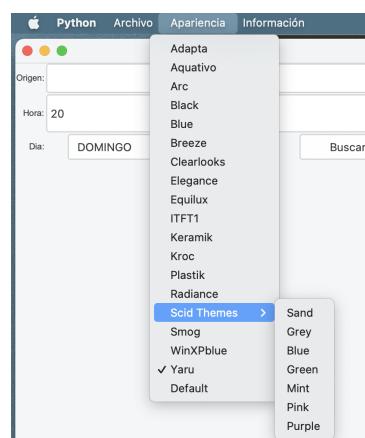
### 7.1.- Nuevo y cerrar

La opción de Nuevo reinicia el programa, vaciando todas las opciones y borrando la ruta anterior, dejándolo en un estado inicial.

Por otro lado, también tenemos la opción de Cerrar, que como su nombre indica cierra el programa.



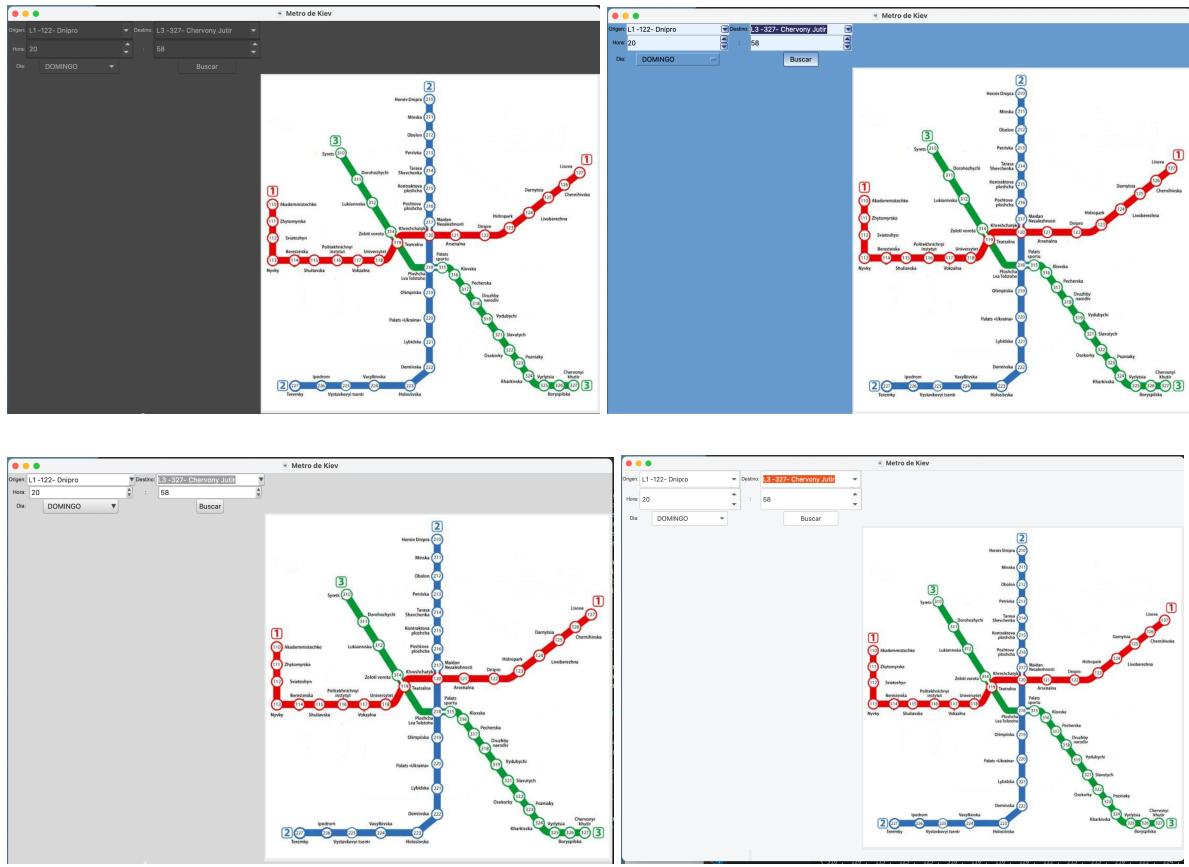
### 7.2.- Apariencia



Nos permite cambiar la interfaz a más de 23 temáticas diferentes gracias al paquete de ttkthemes. A su vez, la temática Scid Themes tiene un submenú con otras subtemáticas.

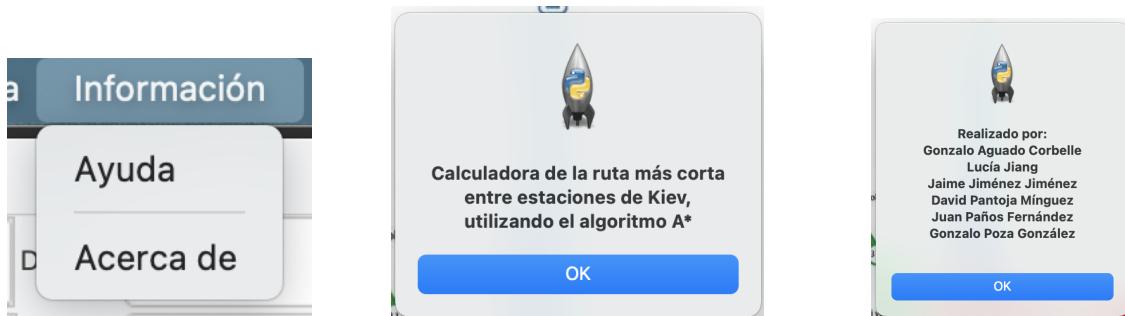
Mencionamos también que esta sección es un Checkbox Menu, que nos permite visualizar cuál es la temática seleccionada.

A continuación mostramos algunos de los temas: Equilux, Blue, Scid Themes (Green) y Yaru



### 7.3.- Información: acerca de e información

Ventanas emergentes que dan una básica de qué hace el programa, y los miembros del grupo.

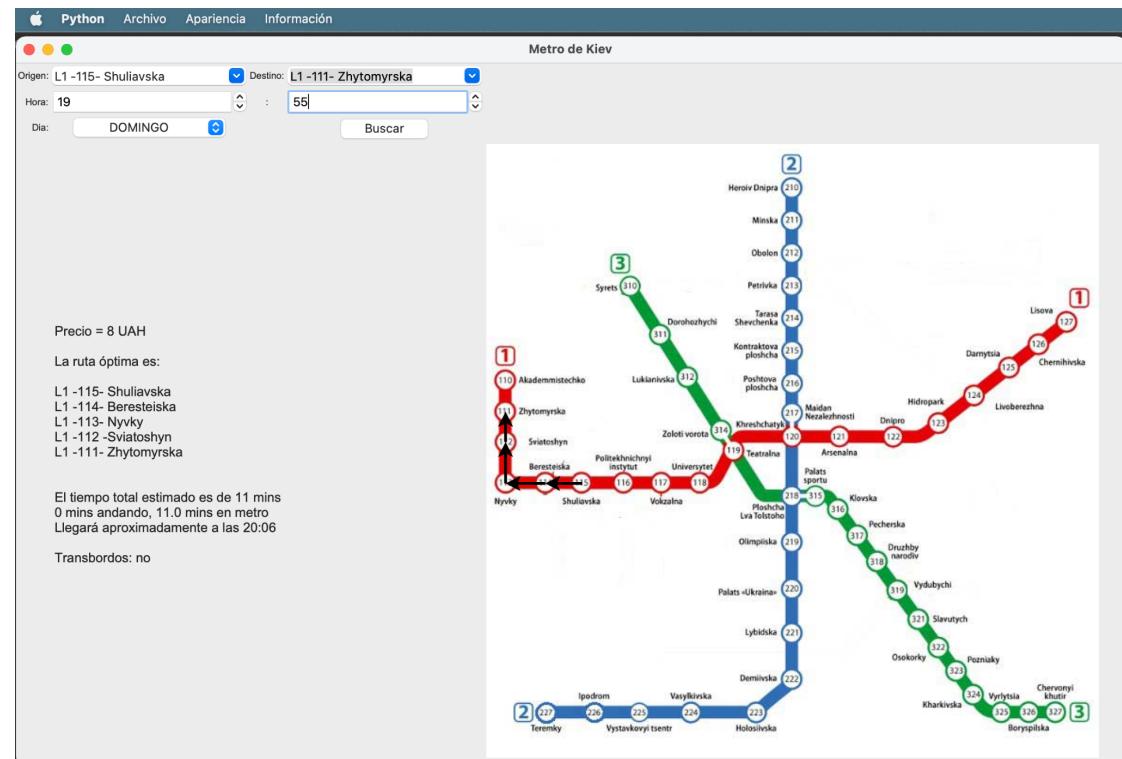


## Visualización de las interfaces en diferentes Sistemas Operativos

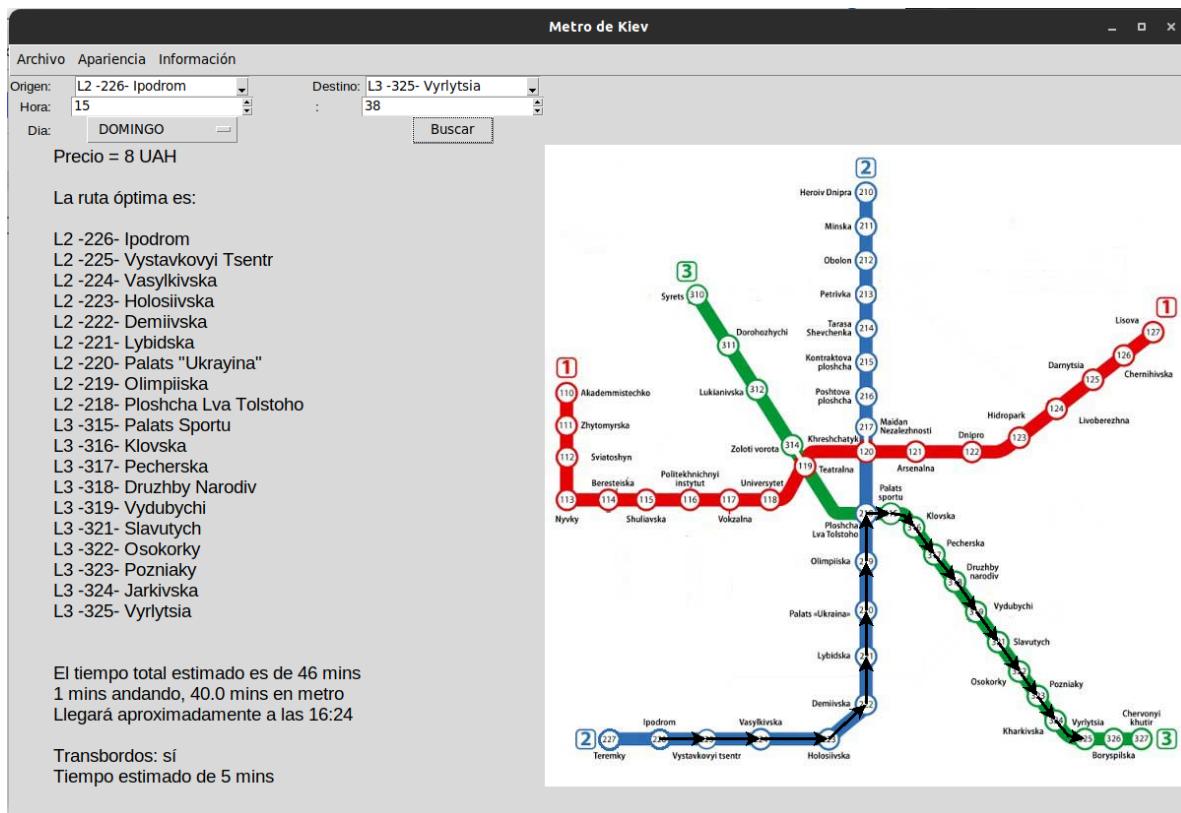
### 1.- Windows:



### 2.- Mac:



### 3.- Linux:



## Problemas y mejoras

### 1.- Problemas surgidos

Uno de los problemas principales que hemos tenido como grupo ha sido la organización para juntarnos todos para realizar el trabajo unidos. Ha habido algunos días en los que no todos los miembros del equipo hemos podido asistir a las reuniones, esto se debe a que la mayor parte del trabajo ha sido realizada en el mes de Noviembre, mes en donde hemos tenido muchos exámenes y entregas. Al final siempre trabajamos todos en alguna parte del trabajo y hemos conseguido terminarlo sin problemas.

En cuanto a problemas del programa en sí, como hemos mencionado previamente, tuvimos que aprender Python desde cero. Por ejemplo, fue una aventura descubrir que Python no tiene un Switches, por lo que hemos tenido que depender de ifs y elifs.

Por otra parte, una de las mayores dificultades ha sido usar la librería ttkthemes. La habíamos visto para mejorar el aspecto de la interfaz del tkinter, pero la instalación en Windows es cuanto menos complicada. No se nos terminaba de aplicar al usar el pip, tuvimos que investigar varias páginas como StackOverflow. Aunque consiguiésemos instalar el paquete de ttkthemes, al final no podíamos importarlo y, aunque programásemos, sólo los compañeros con un sistema operativo Linux o Mac.

Además, no había tanta documentación sobre los ttk como los hay de tkinter, por lo que nos costó entender cómo funcionaba.

## **2.- Mejoras**

Miramos para poner la aplicación como un APK, pero era mucho trabajo ya que tendríamos que haber cambiado la estructura de nuestra interfaz y haber usado otra librería que no habíamos utilizado.

Otra mejora, sería calcular el tiempo de trasbordos en el momento en el que llegas a esa parada en vez de calcularlo al inicio del algoritmo. De esta forma, evitarías pequeños fallos de tiempo en los trasbordos si están cerca de la hora punta. Para mejorar esto último, se podrían obtener los datos en tiempo real del metro de Kiev e introducir el tiempo real hasta que llegue el metro y no una aproximación si el tiempo no es máximo.