MEMORIA
PRÁCTICA GRUPO 1
INTELIGENCIA ARTIFICIAL
METRO DE BUENOS AIRES

Coordinador: Antonio Bielza Díez

[GMI] 5S-1M (24/25) 3er año Grupo 1

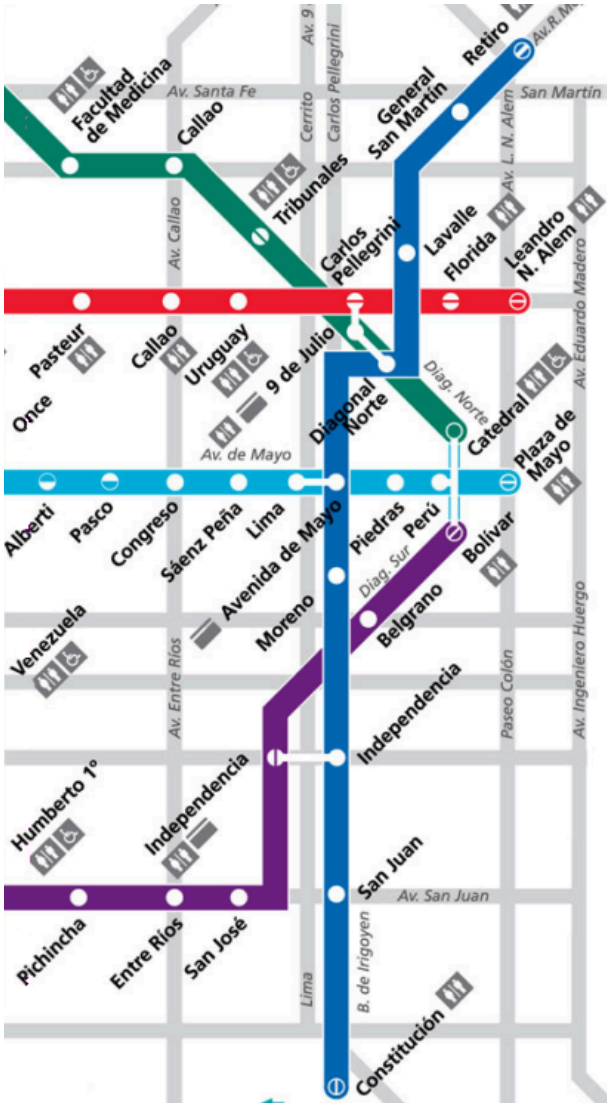
Miembros:
Miguel López García
José David Ortega Yangua
Marco Pérez González
Rodrigo Elola Torrijos
Diego de Arizón Sanz

INTRODUCCIÓN

OBJETIVO

Encontrar el camino óptimo entre dos estaciones en el sistema de Metro de Buenos Aires. Este trayecto será calculado mediante el algoritmo de búsqueda A* y será mostrado usando una aplicación con una interfaz gráfica.

METRO DE BUENOS AIRES



Línea A: Azul Claro
Línea B: Rojo
Línea C: Azul Oscuro
Línea D: Verde Oscuro
Línea E: Púrpura

Transbordos:
Independencia - Independencia
Bolívar - Perú - Catedral
Lima - Avenida de Mayo
Diagonal Norte - 9 de Julio
9 de Julio - Carlos Pellegrini

DESARROLLO DE PRÁCTICA

Durante nuestra fase de diseño planeamos que nuestro algoritmo encontraría el camino más rápido entre las dos estaciones seleccionadas. Para ello tuvimos que crear:

1) Un diccionario con pesos del sistema de metro de Buenos Aires, el cual usa como pesos el tiempo que se tarda de una estación a otra. Las informaciones de tiempos provienen de las estimaciones de tiempos actuales en el Subte de Buenos Aires vía Google Maps, que si bien no corresponden con los de 2009 (año en el que se basa el problema), son los más parecidos que pudimos encontrar.

```python
metro_times = {
    "A": {
        "Plaza de Mayo - Perú": 2,
        "Perú - Piedras": 2,
        "Piedras - Lima": 2,
        "Lima - Sáenz Peña": 2,
        "Sáenz Peña - Congreso": 2,
        "Congreso - Pasco": 2,
        "Pasco - Alberti": 2,
    },
    "B": {
        "Leandro N. Alem - Florida": 2,
        "Florida - Carlos Pellegrini": 2,
        "Carlos Pellegrini - Uruguay": 2,
        "Uruguay - Callao Norte": 2,
        "Callao Norte - Pasteur": 2,
    },
    "C": {
        "Retiro - General San Martín": 2,
        "General San Martín - Lavalle": 3,
        "Lavalle - Diagonal Norte": 2,
        "Diagonal Norte - Avenida de Mayo": 3,
        "Avenida de Mayo - Moreno": 2,
        "Moreno - Independencia Este": 2,
        "Independencia Este - San Juan": 2,
        "San Juan - Constitución": 2,
    },
    "D": {
        "Catedral - 9 de Julio": 2,
        "9 de Julio - Tribunales": 2,
        "Tribunales - Callao Sur": 3,
        "Callao Sur - Facultad de Medicina": 2,
    },
    "E": {
        "Bolívar - Belgrano": 3,
        "Belgrano - Independencia Oeste": 3,
        "Independencia Oeste - San José": 3,
        "San José - Entre Ríos": 3,
        "Entre Ríos - Pichincha": 2,
    },
    "T": {
        "Lima - Avenida de Mayo": 6,
        "Perú - Catedral": 3,
        "Perú - Bolívar": 4,
        "Carlos Pellegrini - 9 de Julio": 3,
        "Diagonal Norte - 9 de Julio": 2,
        "Independencia Oeste - Independencia Este": 4,
        "Catedral - Bolívar": 6,
        "Independencia Este - Independencia Oeste": 2
    },
}
```

```python
def create_network():
    # Create an undirected graph
    G = nx.DiGraph()

    # Add nodes (stations) and edges (connections)
    for line_name, stations in lines.items():
        line_colors = {
            'A': 'light blue',
            'B': 'red',
            'C': 'blue',
            'D': 'green',
            'E': 'purple',
        }

        # Add nodes and edges for each line
        for i in range(len(stations)):
            # Add node with attributes
            G.add_node(stations[i], line=line_name)

            # Add edge to next station if it exists
            if i < len(stations) - 1:
                edge_name = f'{stations[i]} - {stations[i+1]}'
                weight = metro_times[line_name][edge_name]

                G.add_edge(stations[i], stations[i + 1],
                           line=line_name,
                           color=line_colors[line_name],
                           weight=weight)
                G.add_edge(stations[i+1], stations[i],
                           line=line_name,
                           color=line_colors[line_name],
                           weight=weight)

    for transfer, weight in metro_times['T'].items():
        station1, station2 = transfer.split(' - ')
        G.add_edge(station1, station2, line='T', color='orange', weight=weight)
        G.add_edge(station2, station1, line='T', color='orange', weight=weight)

    return G
```

En estas capturas se ve nuestra base de datos con los tiempos entre estaciones separadas por su línea, siendo T el tiempo entre los transbordos y en la otra imagen está la creación de nuestro grafo representado las líneas de metro.

2) Una función heurística h(n) basada en la distancia geodésica entre las estaciones del plano

```python
metro_coord_geodesic = {
    'Plaza de Mayo': (-34.60852336525573, -58.37099647736891),
    'Perú': (-34.60841740052505, -58.37424731475452),
    'Piedras': (-34.60868231209738, -58.3782706282344),
    'Lima': (-34.60899936126676, -58.38223798242305),
    'Sáenz Peña': (-34.60930642035077, -58.38642070055631),
    'Congreso': (-34.60909205349264, -58.39239405734514),
    'Pasco': (-34.60943212397035, -58.398124817661504),
    'Alberti': (-34.609737510219716, -58.40065898341956),

    'Leandro N. Alem': (-34.60281575311817, -58.369699552240384),
    'Florida': (-34.603168992189694, -58.374205663467976),
    'Carlos Pellegrini': (-34.603515609413634, -58.38091914892496),
    'Uruguay': (-34.603876515306226, -58.38648778697883),
    'Callao Norte': (-34.60428051748904, -58.391901606970926),
    'Pasteur': (-34.60447816473591, -58.39891966425801),
```

```python
    'Retiro': (-34.59102559582136, -58.37459659456844),
    'General San Martín': (-34.59542282799334, -58.37703420805947),
    'Lavalle': (-34.60196300063525, -58.37785934938813),
    'Diagonal Norte': (-34.60421016183624, -58.37767915586964),
    'Avenida de Mayo': (-34.60836625694821, -58.37838785470674),
    'Moreno': (-34.61191577252951, -58.379483040138766),
    'Independencia Este': (-34.617505429647466, -58.37847215489829),
    'San Juan': (-34.621956046240555, -58.37929951603437),
    'Constitución': (-34.6273143709629, -58.380378469073364),

    'Catedral': (-34.60746965446476, -58.373881894567646),
    '9 de Julio': (-34.604374394134, -58.37987593524887),
    'Tribunales': (-34.60167148679957, -58.384128779029666),
    'Callao Sur': (-34.59949557978796, -58.39195888279371),
    'Facultad de Medicina': (-34.59969268410233, -58.39749535538687),

    'Bolívar': (-34.609429858008006, -58.3735886638776),
    'Belgrano': (-34.61270583182017, -58.37760124862716),
    'Independencia Oeste': (-34.618041135928785, -58.38121954864429),
    'San José': (-34.62220323205144, -58.384870612201205),
    'Entre Ríos': (-34.62261344500609, -58.39116573363126),
    'Pichincha': (-34.623049298114374, -58.3967793357254),
}
```

```python
def heuristic(node, target):
    coord_node = metro_coord_geodesic[node]
    coord_target = metro_coord_geodesic[target]
    return geopy.distance.geodesic(coord_node,coord_target).km
```

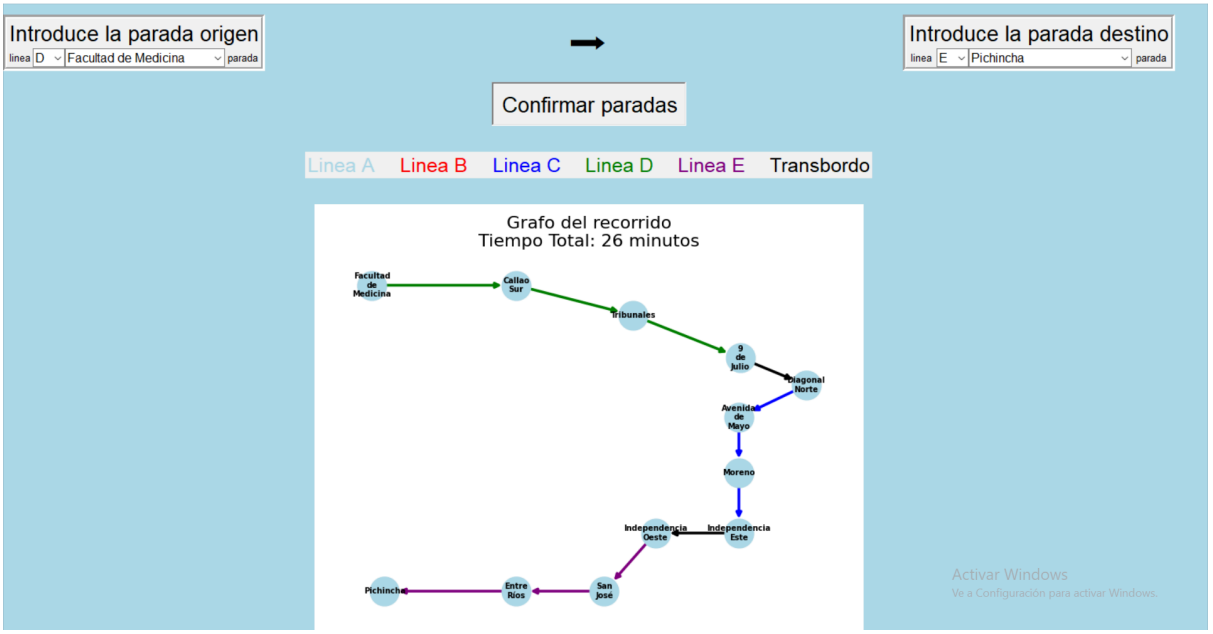
En estas capturas se observan las coordenadas geodésicas de nuestras estaciones en el plano así como la función heurística que calcula la distancia geodésica entre cualquier par de estaciones.

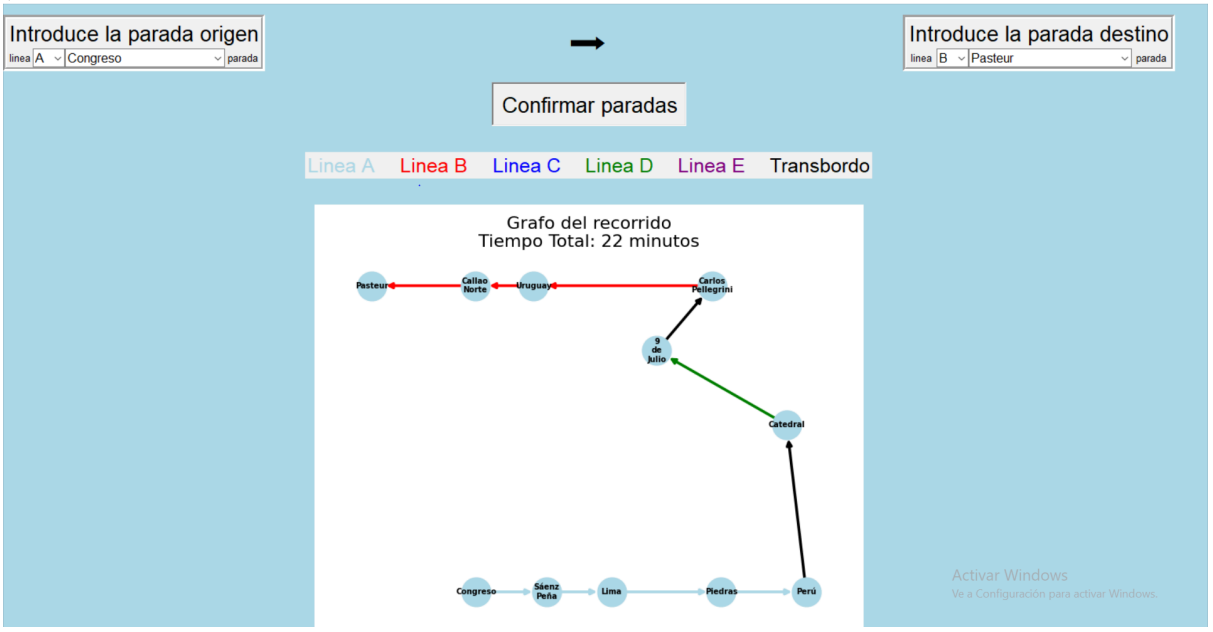
3) La función g(n) en A* se calcula el coste acumulado, es decir, la suma de los pesos de las aristas que forman parte del recorrido hasta la estación n. Al estar g(n) implementado dentro de networkx.astar las capturas son del archivo astar.py creado al instalar networkx

```python
for neighbor, w in G_succ[curnode].items():
    cost = weight(curnode, neighbor, w)
    if cost is None:
        continue
    ncost = dist + cost
```

```
Notes
-----
Edge weight attributes must be numerical.
Distances are calculated as sums of weighted edges traversed.
```

4) Una interfaz gráfica para el uso de un futuro usuario de esta aplicación. En nuestra interfaz gráfica hay que seleccionar una línea de metro y una estación de ella tanto para inicio como para destino de nuestro trayecto, al confirmar estos datos se nos mostrará un mapa del camino recorrido indicando el tiempo total del viaje y la línea en la que se viaja en cada momento mediante el uso de colores.





Las posiciones de los nodos mostrados en la interfaz gráfica siguen estas coordenadas en vez de las coordenadas geodésicas en beneficio de la legibilidad del grafo del recorrido.

```python
metro_node_coord = {
    'Plaza de Mayo': (4.2,0.7),
    'Perú': (3,0.7),
    'Piedras': (1.69,0.7),
    'Lima': (0,0.7),
    'Sáenz Peña': (-1,0.7),
    'Congreso': (-2.08,0.7),
    'Pasco': (-3.23,0.7),
    'Alberti': (-4.26,0.7),

    'Leandro N. Alem': (3.95,2.35),
    'Florida': (2.75,2.35),
    'Carlos Pellegrini': (1.55,2.35),
    'Uruguay': (-1.2,2.35),
    'Callao Norte': (-2.1,2.35),
    'Pasteur': (-3.69,2.35),

    'Retiro': (3.83,4.63),
    'General San Martín': (2.8,4.1),
    'Lavalle': (1.89,2.82),
    'Diagonal Norte': (1.52,1.4),
    'Avenida de Mayo': (0.68,0.69),
    'Moreno': (0.68,-0.53),
    'Independencia Este': (0.68,-1.86),
    'San Juan': (0.68,-3.36),
    'Constitución': (0.68,-4.88),

    'Catedral': (2.7,1.6),
    '9 de Julio': (0.7,2),
    'Tribunales': (-0.64,2.94),
    'Callao Sur': (-2.1,3.6),
    'Facultad de Medicina': (-3.9,3.6),

    'Bolívar': (2.7,-0.15),
    'Belgrano': (1.2,-0.57),
    'Independencia Oeste': (-0.36,-1.86),
    'San José': (-1.0,-3.14),
    'Entre Ríos': (-2.1,-3.14),
    'Pichincha': (-3.74,-3.14),
}
```

```python
import tkinter as tk
from tkinter import messagebox, ttk

import networkx as nx

from matplotlib import pyplot as plt
from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg

import time as t
import sys

from map_constants import metro_times, lines, metro_node_coord
from map_import astar


def Window(G):

    # Funciones para cuando cambie el elemento de un desplegable
    def origen_changed(event):
        comboOrigen["values"] = lines[comboLineaOrigen.get()]
        comboOrigen['state'] = 'normal'

    def destino_changed(event):
        comboDestino["values"] = lines[comboLineaDestino.get()]
        comboDestino['state'] = 'normal'

    # Definicion de la ventana para la GUI
    window = tk.Tk()
    #Le ponemos nombre a la ventana
    window.title("Camino mas corto metro Buenos Aires")
    #Configuramos la disposicion de los objetos
    window.rowconfigure([0, 2], minsize=100)
    window.columnconfigure([0, 2], minsize=623)
    #Para que la pantalla salga maximizada al abrirla
    window.state('zoomed')
    window.configure(bg='lightblue')

    # Definimos el espacio para introducir la estacion origen
    frameOrigen = tk.Frame(master=window, relief=tk.GROOVE, borderwidth=5)
    labelOrigen = tk.Label(text="Introduce la parada origen", font=("Arial", 20), master=frameOrigen)
    labelOrigen.pack(expand=True, side=tk.TOP)
    # Definimos el espacio para introducir la estacion origen
    # Labels para linea y parada
    lblLineaOrigen = tk.Label(master=frameOrigen, text="linea", font=("Arial", 10))
    lblParadaOrigen = tk.Label(master=frameOrigen, text="parada", font=("Arial", 10))
    lblLineaOrigen.pack(expand=True, side=tk.LEFT)
    lblParadaOrigen.pack(expand=True, side=tk.RIGHT)

    # Dropdowns para elegir la linea y parada origen
    comboOrigen = ttk.Combobox(master=frameOrigen, state="readonly", font='Arial 12', values=["Escoja primero la linea"])
    comboLineaOrigen = ttk.Combobox(master=frameOrigen, width=2, state="readonly", font='Arial 12', values=["A", "B", "C", "D", "E"])
    comboLineaOrigen.bind("<<ComboboxSelected>>", origen_changed)
    comboLineaOrigen.pack(fill=tk.BOTH, side=tk.LEFT)
    comboOrigen.pack(fill=tk.BOTH, expand=True, side=tk.RIGHT)

    # Creamos el frame contenedor de los elementos del destino
    frameDestino = tk.Frame(master=window, relief=tk.GROOVE, borderwidth=5)
    labelDestino = tk.Label(text="Introduce la parada destino", font=("Arial", 20), master=frameDestino)
    labelDestino.pack(expand=True)

    # Definimos el espacio para introducir la estacion destino
    # Labels para linea y parada
    lblLineaDestino = tk.Label(master=frameDestino, text="linea", font=("Arial", 10))
```

```python
    lblParadaDestino = tk.Label(master=frameDestino, text="parada", font=("Arial", 10))
    lblLineaDestino.pack(expand=True, side=tk.LEFT)
    lblParadaDestino.pack(expand=True, side=tk.RIGHT)

    # Dropdowns para elegir la linea y parada destino
    comboDestino = ttk.Combobox(master=frameDestino, state="readonly", font='Arial 12', values=["Escoja primero la linea"])
    comboLineaDestino = ttk.Combobox(master=frameDestino, width=2, state="readonly", font='Arial 12', values=["A", "B", "C", "D", "E"])
    comboLineaDestino.bind("<<ComboboxSelected>>", destino_changed)
    comboLineaDestino.pack(fill=tk.BOTH, side=tk.LEFT)
    comboDestino.pack(fill=tk.BOTH, expand=True, side=tk.RIGHT)

    # Definimos el espacio para contener la flecha de relacion origen-destino
    frameFlecha = tk.Frame(master=window, relief=tk.FLAT, borderwidth=5, background='lightblue')
    labelFlecha = tk.Label(master=frameFlecha, font=("Arial", 40), text="\u2B95", background='lightblue')
    labelFlecha.pack(expand=True)

    # Posicionamiento de los frames
    frameOrigen.grid(row=0, column=0, sticky="W")
    frameDestino.grid(row=0, column=2, sticky="E")
    frameFlecha.grid(row=0, column=1)

    def comprobarLineasElegidas():
        if comboLineaOrigen.current() == -1:
            raise ValueError("No se ha seleccionado la linea origen")
        elif comboLineaDestino.current() == -1:
            raise ValueError("No se ha seleccionado la linea destino")
        elif comboOrigen.current() == -1:
            raise ValueError("La estación de origen no es valida")
        elif comboDestino.current() == -1:
            raise ValueError("La estacion de destino no es valida")

    # pathImage se ejecuta cuando se pulsa el boton de confirmar paradas
    # Comprueba que efectivamente se han escogido las lineas y paradas
    # Crea el canvas que será el encargado de guardar el grafo de las estaciones que hay que pasar
    # Crea también el grafo y realiza con una llamada a la funcion map.astar() el camino mínimo
    # Se determina como tiempo mínimo del tren en una parada 1 minuto
    def pathImage(event):
        try:
            confirmBtn["relief"] = tk.SUNKEN
            comprobarLineasElegidas()
            canvas = FigureCanvasTkAgg(figure=plt.figure(figsize=(6,5)), master = window)
            canvas.get_tk_widget().destroy()
            origen = comboOrigen.get()
            destino = comboDestino.get()
            shortest_path = astar(G, source=origen, target=destino)
            graph = nx.DiGraph()
            time = 0
            for i in range(len(shortest_path)):
                graph.add_node(shortest_path[i].replace(" ", "\n"), pos=metro_node_coord[shortest_path[i]])
                if i < len(shortest_path) - 1:
                    edge = f'{shortest_path[i]} - {shortest_path[i+1]}'
                    edgeInv = f'{shortest_path[i+1]} - {shortest_path[i]}'
                    line = ""
                    weight = ""
                    color = ''
                    for linea in metro_times:
                        if edge in metro_times[linea]:
                            line = linea
                            if linea == "A":
                                color = 'lightblue'
                            elif linea == "B":
                                color = 'r'
```

```python
                            elif linea == "C":
                                color = 'b'
                            elif linea == "D":
                                color = 'g'
                            elif linea == "E":
                                color = 'purple'
                            else:
                                color = 'k'
                            weight = metro_times[linea][edgeInv]
                            time += int(weight)
                            break

                    graph.add_edge(shortest_path[i].replace(" ", "\n"), shortest_path[i+1].replace(" ", "\n"), color=color, line='')

            # Set up the plot
            Figure = plt.figure(figsize=(7, 5.5))
            ax = plt.gca()

            # Draw the graph
            pos = nx.get_node_attributes(graph,'pos')
            edges = graph.edges()
            colors = [graph[u][v]['color'] for u,v in edges]
            nx.draw(graph, pos, ax=ax, with_labels=True, edge_color=colors, node_color='lightblue', node_size=700, font_size=7, font_weight='bold', width=2.5)

            # Add edge labels
            edge_labels = nx.get_edge_attributes(graph, 'line')
            nx.draw_networkx_edge_labels(graph, pos, edge_labels=edge_labels, font_size=6)

            # Set title and remove axis
            plt.title(f'Grafo del recorrido\nTiempo Total: {time} minutos', fontsize=16)
            plt.axis('off')

            # Adjust layout and display
            plt.tight_layout()
            # creating the Tkinter canvas
            # containing the Matplotlib figure
            canvas = FigureCanvasTkAgg(Figure, master = window)
            canvas.draw()

            #Definimos las leyendas para cada linea
            frameLeyenda = tk.Frame(master=window)
            #Leyenda = master, colorTexto, texto, fuente
            leyendaA = tk.Label(master=frameLeyenda, fg='lightblue', text='Linea A\t', font="Arial 19")
            leyendaB = tk.Label(master=frameLeyenda, fg='red', text='Linea B\t', font="Arial 19")
            leyendaC = tk.Label(master=frameLeyenda, fg='blue', text='Linea C\t', font="Arial 19")
            leyendaD = tk.Label(master=frameLeyenda, fg='green', text='Linea D\t', font="Arial 19")
            leyendaE = tk.Label(master=frameLeyenda, fg='purple', text='Linea E\t', font="Arial 19")
            leyendaT = tk.Label(master=frameLeyenda, fg='black', text='Transbordo', font="Arial 19")
            #Las colocamos en orden
            leyendaA.pack(side=tk.LEFT)
            leyendaB.pack(side=tk.LEFT)
            leyendaC.pack(side=tk.LEFT)
            leyendaD.pack(side=tk.LEFT)
            leyendaE.pack(side=tk.LEFT)
            leyendaT.pack(side=tk.LEFT)
            frameLeyenda.grid(row=2, column=0, columnspan=3)

            # placing the canvas on the Tkinter window
            canvas.get_tk_widget().grid(row=3, column=0, columnspan=3)
```

```python
            # Eliminamos la figura para liberar espacio en memoria
            # Se pasa el argumento all por si había quedado alguna figura abierta, aunque no debería pasar
            plt.close('all')


            t.sleep(0.1)
            confirmBtn["relief"] = tk.RAISED
        # Tratamiento de los errores que pueda generar la funcion
        # ValueError será normalmente un error que hayamos subido nosotros
        # Exception es un valor generico para todas las demas excepciones
        except ValueError as err:
            confirmBtn["relief"] = tk.RAISED
            messagebox.showerror(title="ERROR", message=err)
        except Exception as err:
            confirmBtn["relief"] = tk.RAISED
            messagebox.showerror(title="ERROR", message=f'Error del tipo: {type(err)}\nTraza del error: {err}')

    # Definimos el boton y su Frame contenedor
    frameBtn = tk.Frame(master=window, width=50)
    confirmBtn = tk.Button(master=frameBtn, text="Confirmar paradas", font="Arial 20", relief=tk.RAISED)
    confirmBtn.bind("<Button-1>", pathImage)
    confirmBtn.pack(side=tk.BOTTOM)
    frameBtn.grid(row=1, column=1)

    # Funcion para regular como se comporta la aplicacion al cerrar la ventana
    def windowClosed():
        # Eliminamos la figura para liberar espacio en memoria
        # Se pasa el argumento all por si había quedado alguna figura abierta, aunque no debería pasar
        plt.close('all')
        window.destroy()
        sys.exit()

    # Asociamos la accion de cerrar la ventana
    window.protocol("WM_DELETE_WINDOW", windowClosed)

    # Main loop de la ventana (Hecho por defecto)
    window.mainloop()
```

Aquí vemos la implementación de la GUI que realizamos usando tkinter , como se configuraron los desplegables y la construcción del mapa visual con todo el rango de colores.

CONCLUSIONES TÉCNICAS

En general tuvimos un grupo muy bueno ya que todos participamos activamente y apoyamos a la elaboración de este proyecto, además tuvimos una buena comunicación a la hora de transmitir las dudas o problemas .

De la parte técnica las mayores dificultades fueron el uso de Python, ya que varios de los miembros no estaban familiarizados con este lenguaje y la creación de la interfaz gráfica que conlleva mucho trabajo.

Afortunadamente, de estos obstáculos obtuvimos muchos conocimientos en el lenguaje, además de saber cómo implementar y utilizar los algoritmos de búsqueda vistos en clase, en este caso específico el A*.

REFERENCIAS

FUENTES

Google Maps para los tiempos de trayectos entre estaciones.https://www.google.com/maps

LIBRERÍAS

networkx: Utilizada para la creación y definición del grafo y la ejecución del algoritmo A*.
https://networkx.org/documentation/stable/reference/index.html

geopy: Usada para el cálculo de las distancias geodésicas de la función heurística.
https://geopy.readthedocs.io/en/stable/#module-geopy.distance

tkinter: Usada en la interfaz gráfica.https://docs.python.org/es/3/library/tkinter.html

matplotlib: Usada en la interfaz gráfica.https://matplotlib.org/stable/index.html