

## ✓ Actividad 2 - Búsqueda y sistemas basados en reglas

Conforme a la actividad, se realizará un **Sistema Experto para transporte masivo**

El proyecto consistirá en desarrollar un sistema que determine la mejor ruta en un sistema de transporte masivo local utilizando datos cargados desde un archivo CSV

### 1. Crear y Preparar el Archivo

El archivo CSV contiene los siguientes datos con las siguientes columnas:

- Ciudad\_Origen: Nombre de la estación de inicio.
- Ciudad\_Destino: Nombre de la estación de destino.
- Distancia\_km: Distancia entre las estaciones (en kilómetros).
- Tiempo\_min: Tiempo estimado para viajar entre las estaciones (en minutos).
- Costo\_COP: Costo del viaje en pesos colombianos.
- Transbordos: Número de transbordos requeridos.
- Disponible: Indica si la ruta está disponible

### 2. Cargar los Datos desde el Archivo

Utilizaremos pandas para cargar el archivo rutas.csv en un DataFrame y trabajar con los datos dinámicamente.

```
import pandas as pd

raw_url = 'https://raw.githubusercontent.com/feliperamon1/Personal/main/rutas.csv'

# Cargar el archivo CSV usando el raw URL
rutas_df = pd.read_csv(raw_url)

# Mostrar los datos cargados
print("Datos de rutas cargados desde el archivo CSV:")
print(rutas_df)
```

↗ Datos de rutas cargados desde el archivo CSV:

	Ciudad_Origen	Ciudad_Destino	Distancia_km	Tiempo_min	Costo_COP	\
0	Estación A	Estación B	5	10	2000	
1	Estación A	Estación C	10	25	3000	
2	Estación B	Estación C	7	15	1500	
3	Estación B	Estación D	12	30	2500	
4	Estación C	Estación D	8	20	2000	
5	Estación C	Estación E	15	40	3500	
6	Estación D	Estación E	20	50	4000	

	Transbordos	Disponible
0	0	True
1	1	True
2	0	True
3	1	True
4	1	False
5	2	True
6	2	True

### 3. Definir el Sistema Experto

Usamos el framework Experta para implementar la lógica del sistema experto.

a. Definimos los hechos:

- Ruta: Representa cada posible conexión entre dos estaciones.
- Preferencias: Representa las preferencias del usuario (tiempo, costo, transbordos).

b. Definimos las reglas:

- Recomendaciones basadas en tiempo, costo, transbordos y disponibilidad.

```
pip install experta
```

↗ [Mostrar el resultado oculto](#)

```
!python3 -m venv mi_entorno
!source mi_entorno/bin/activate
!pip install experta==1.9.4
```

 [Mostrar el resultado oculto](#)

```
!pip install --upgrade frozendict
```

 [Mostrar el resultado oculto](#)

```
from experta import *

class Ruta(Fact):
    """
    Representa una ruta entre dos estaciones en el sistema de transporte masivo.
    """
    pass

class Preferencias(Fact):
    """
    Representa las preferencias del usuario para seleccionar la mejor ruta.
    """
    pass

class SistemaTransporteMasivo(KnowledgeEngine):
    """
    Sistema experto que evalúa y recomienda rutas en el sistema de transporte masivo.
    """

    @Rule(
        Ruta(origen=MATCH.origen, destino=MATCH.destino, tiempo=P(lambda x: x <= 20)),
        Preferencias(prioridad='tiempo')
    )
    def ruta_rapida(self, origen, destino):
        print(f"Recomendación: Use la ruta más rápida de {origen} a {destino} (<= 20 minutos).")

    @Rule(
        Ruta(origen=MATCH.origen, destino=MATCH.destino, costo=P(lambda x: x <= 3000)),
        Preferencias(prioridad='costo')
    )
    def ruta_economica(self, origen, destino):
        print(f"Recomendación: Use la ruta más económica de {origen} a {destino} (<= 3000 COP).")

    @Rule(
        Ruta(origen=MATCH.origen, destino=MATCH.destino, transbordos=P(lambda x: x <= 1)),
        Preferencias(prioridad='comodidad')
    )
    def ruta_con_menos_transbordos(self, origen, destino):
        print(f"Recomendación: Use la ruta con menos transbordos de {origen} a {destino} (<= 1 transbordo).")

    @Rule(
        Ruta(origen=MATCH.origen, destino=MATCH.destino, disponible=False)
    )
    def ruta_no_disponible(self, origen, destino):
        print(f"Advertencia: La ruta de {origen} a {destino} no está disponible actualmente.")
```

#### 4. Configurar y Ejecutar el Sistema

##### a. Cargar datos y crear el motor:

- Declaramos cada ruta como un hecho.
- Declaramos las preferencias del usuario.

##### b. Ejecutar reglas:

- Evaluamos cada ruta según las preferencias dadas.


```
# Crear el motor
motor = SistemaTransporteMasivo()

# Evaluar rutas cargadas desde el archivo CSV
for _, ruta in rutas_df.iterrows():
```

```

motor.reset()
motor.declare(Ruta(
    origen=ruta['Ciudad_Origen'],
    destino=ruta['Ciudad_Destino'],
    tiempo=ruta['Tiempo_min'],
    costo=ruta['Costo_COP'],
    transbordos=ruta['Transbordos'],
    disponible=ruta['Disponible']
))
motor.declare(Preferencias(prioridad='tiempo')) # Cambia 'tiempo' a 'costo' o 'comodidad' según el caso
motor.run()

```

 Recomendación: Use la ruta más rápida de Estación A a Estación B ( $\leq 20$  minutos).  
 Recomendación: Use la ruta más rápida de Estación B a Estación C ( $\leq 20$  minutos).  
 Recomendación: Use la ruta más rápida de Estación C a Estación D ( $\leq 20$  minutos).  
 Advertencia: La ruta de Estación C a Estación D no está disponible actualmente.

## 5. Visualización Gráfica

Utilizamos NetworkX para representar las rutas gráficamente.

```

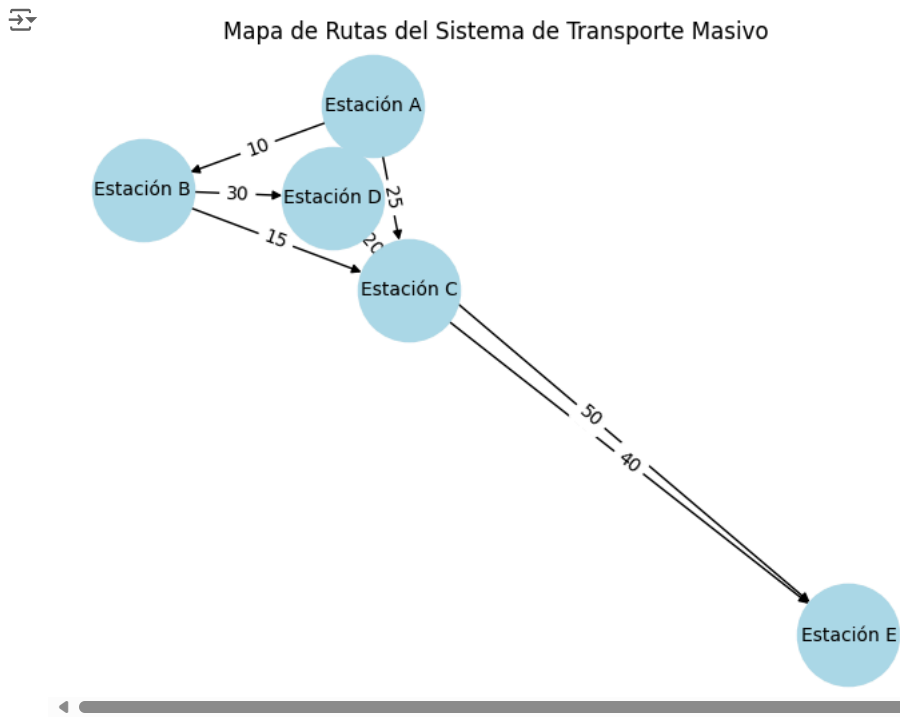
import networkx as nx
import matplotlib.pyplot as plt

def graficar_rutas(rutas_df):
    G = nx.DiGraph()
    for _, row in rutas_df.iterrows():
        G.add_edge(row['Ciudad_Origen'], row['Ciudad_Destino'], weight=row['Tiempo_min'])

    pos = nx.spring_layout(G)
    nx.draw(G, pos, with_labels=True, node_size=3000, node_color="lightblue", font_size=10)
    edge_labels = nx.get_edge_attributes(G, 'weight')
    nx.draw_networkx_edge_labels(G, pos, edge_labels=edge_labels)
    plt.title("Mapa de Rutas del Sistema de Transporte Masivo")
    plt.show()

# Visualizar rutas
graficar_rutas(rutas_df)

```



## 6. Implementación de un Algoritmo de Optimización (Dijkstra)

Usaremos NetworkX para encontrar la ruta más corta basándonos en tiempo o costo.

```
import networkx as nx

def construir_grafo(rutas_df, criterio='Tiempo_min'):
    """
    Construye un grafo dirigido a partir de las rutas y elige un criterio como peso.
    """
    G = nx.DiGraph()
    for _, row in rutas_df.iterrows():
        if row['Disponible']: # Solo incluir rutas disponibles
            G.add_edge(row['Ciudad_Origen'], row['Ciudad_Destino'], weight=row[criterio])
    return G

def encontrar_mejor_ruta(origen, destino, grafo):
    """
    Encuentra la mejor ruta entre origen y destino usando el grafo.
    """
    try:
        ruta = nx.shortest_path(grafo, source=origen, target=destino, weight='weight')
        costo = nx.shortest_path_length(grafo, source=origen, target=destino, weight='weight')
        return ruta, costo
    except nx.NetworkXNoPath:
        return None, None

# Construir el grafo
grafo = construir_grafo(rutas_df, criterio='Tiempo_min')

# Encontrar la mejor ruta
origen, destino = "Estación A", "Estación E"
ruta, costo = encontrar_mejor_ruta(origen, destino, grafo)
if ruta:
    print(f"Mejor ruta basada en tiempo: {' -> '.join(ruta)} con un costo de {costo} minutos")
else:
    print(f"No se encontró una ruta entre {origen} y {destino}.")
```

➡ Mejor ruta basada en tiempo: Estación A -> Estación C -> Estación E con un costo de 65 minutos

## 7. Visualización del Algoritmo Dijkstra

Usaremos NetworkX y Matplotlib para dibujar el grafo con la ruta más corta resaltada.

```
import matplotlib.pyplot as plt

# Construcción del Grafo
def construir_grafo(rutas_df, criterio='Tiempo_min'):
    """
    Construye un grafo dirigido a partir de las rutas y elige un criterio como peso.
    """
    G = nx.DiGraph()
    for _, row in rutas_df.iterrows():
        if row['Disponible']: # Solo incluir rutas disponibles
            G.add_edge(row['Ciudad_Origen'], row['Ciudad_Destino'], weight=row[criterio])
    return G

# Visualización del Grafo y Resaltado de la Ruta Óptima
def visualizar_ruta_dijkstra(origen, destino, grafo):
    """
    Visualiza el grafo completo y resalta la ruta más corta calculada con Dijkstra.
    """
    try:
        # Ruta más corta basada en Dijkstra
        ruta_optima = nx.shortest_path(grafo, source=origen, target=destino, weight='weight')
        peso_total = nx.shortest_path_length(grafo, source=origen, target=destino, weight='weight')

        # Configuración del grafo para la visualización
        pos = nx.spring_layout(grafo)
        plt.figure(figsize=(10, 8))

        # Dibujar el grafo completo
        nx.draw(grafo, pos, with_labels=True, node_size=3000, node_color="lightgray", font_size=10, font_weight="bold")

        # Resaltar la ruta óptima
        edges_optima = list(zip(ruta_optima[:-1], ruta_optima[1:]))
        nx.draw_networkx_edges(grafo, pos, edgelist=edges_optima, edge_color='red', width=2)

        # Añadir etiquetas de peso
```

```

edge_labels = nx.get_edge_attributes(grafo, 'weight')
nx.draw_networkx_edge_labels(grafo, pos, edge_labels=edge_labels)

# Mostrar resultados
plt.title(f"Ruta óptima: {' -> '.join(ruta_optima)} (Peso total: {peso_total})", fontsize=14)
plt.show()

except nx.NetworkXNoPath:
    print(f"No se encontró una ruta entre {origen} y {destino}.")

# Crear datos simulados de rutas
import pandas as pd

rutas_df = pd.DataFrame({
    'Ciudad_Origen': ['Estación A', 'Estación A', 'Estación B', 'Estación B', 'Estación C', 'Estación C', 'Estación D'],
    'Ciudad_Destino': ['Estación B', 'Estación C', 'Estación C', 'Estación D', 'Estación D', 'Estación E', 'Estación E'],
    'Distancia_km': [5, 10, 7, 12, 8, 15, 20],
    'Tiempo_min': [10, 25, 15, 30, 20, 40, 50],
    'Costo_COP': [2000, 3000, 1500, 2500, 2000, 3500, 4000],
    'Transbordos': [0, 1, 0, 1, 1, 2, 2],
    'Disponibile': [True, True, True, True, False, True, True]
})

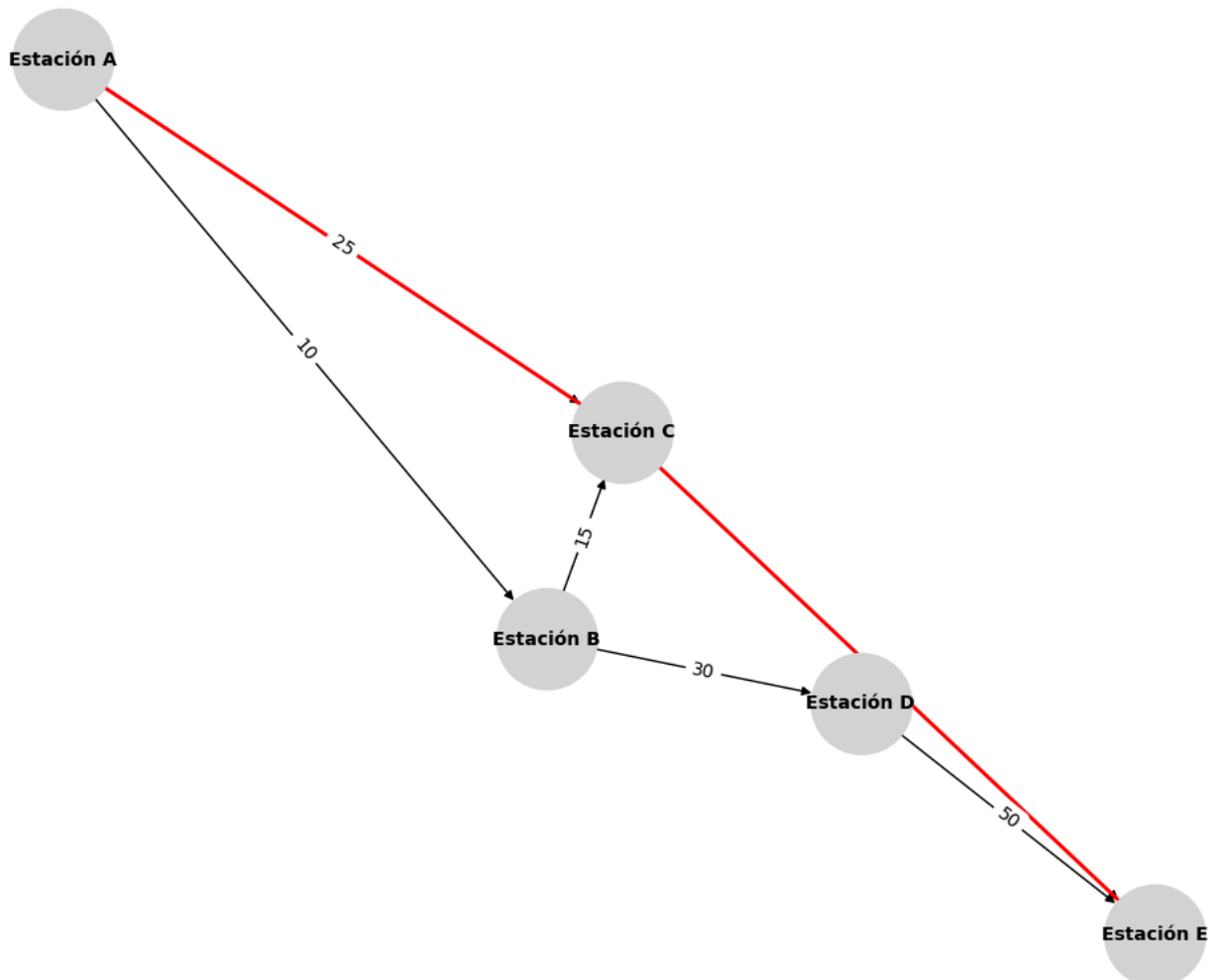
# Construir el grafo
grafo = construir_grafo(rutas_df, criterio='Tiempo_min')

# Visualizar la ruta óptima entre dos estaciones
visualizar_ruta_dijkstra("Estación A", "Estación E", grafo)

```



Ruta óptima: Estación A -> Estación C -> Estación E (Peso total: 65)



Desarrollaremos una interfaz gráfica para que el usuario pueda seleccionar su origen, destino y preferencia de criterio (tiempo o costo).

```
import ipywidgets as widgets
from IPython.display import display
import networkx as nx
import pandas as pd

# Crear datos simulados de rutas
rutas_df = pd.DataFrame({
    'Ciudad_Origen': ['Estación A', 'Estación A', 'Estación B', 'Estación B', 'Estación C', 'Estación C', 'Estación D'],
    'Ciudad_Destino': ['Estación B', 'Estación C', 'Estación C', 'Estación D', 'Estación D', 'Estación E', 'Estación E'],
    'Tiempo_min': [10, 25, 15, 30, 20, 40, 50],
    'Costo_COP': [2000, 3000, 1500, 2500, 2000, 3500, 4000],
    'Disponible': [True, True, True, True, False, True, True]
})

# Construcción del grafo
def construir_grafo(rutas_df, criterio='Tiempo_min'):
    """
    Construye un grafo dirigido a partir de las rutas y elige un criterio como peso.
    """
    G = nx.DiGraph()
    for _, row in rutas_df.iterrows():
        if row['Disponible']:
            G.add_edge(row['Ciudad_Origen'], row['Ciudad_Destino'], weight=row[criterio])
    return G

grafo = construir_grafo(rutas_df)


# Listas desplegables con opciones dinámicas
origen = widgets Dropdown(options=rutas_df['Ciudad_Origen'].unique(), description="Origen:")
destino = widgets Dropdown(options=rutas_df['Ciudad_Destino'].unique(), description="Destino:")
criterio = widgets Dropdown(options=["Tiempo_min", "Costo_COP"], description="Criterio:")
boton = widgets.Button(description="Buscar Ruta")

# Resultado
resultado = widgets.Output()

# Acción del botón
def buscar_ruta(b):
    with resultado:
        resultado.clear_output()
        try:
            ruta_optima = nx.shortest_path(grafo, source=origen.value, target=destino.value, weight='weight')
            peso_total = nx.shortest_path_length(grafo, source=origen.value, target=destino.value, weight='weight')
            print(f"Mejor ruta: {' -> '.join(ruta_optima)}")
            print(f"Peso total ({criterio.value}): {peso_total}")
        except nx.NetworkXNoPath:
            print("No se encontró una ruta válida entre las estaciones seleccionadas.")

boton.on_click(buscar_ruta)

# Mostrar widgets
display(origen, destino, criterio, boton, resultado)
```



Origen:

Estación A

Destino:

Estación D

Criterio:

Tiempo\_min

Buscar Ruta

Mejor ruta: Estación A -> Estación B -> Estación D

Peso total (Tiempo min): 40

