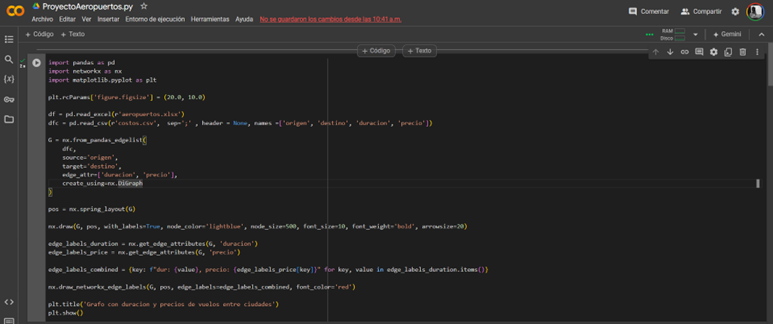
Documentación código Estructura de datos y algoritmos 1

Inicialmente creamos nuestro código en visual studio code debido a la interfaz amigable y su facilidad de manejo con respecto a las demás, luego pasamos el código a la plataforma online Colab para lograr ver el resultado de la gráfica del grafo, como se muestra a continuación: (Ver imagen 1, 2)

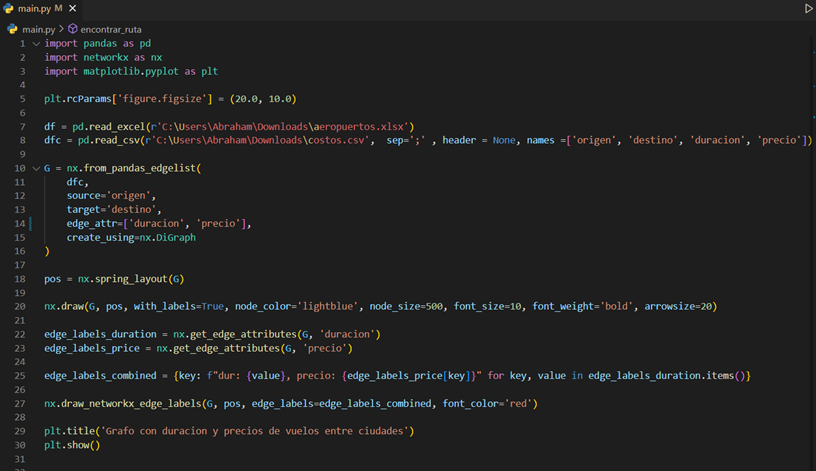
Colaboratory

Imagen 1



Visual Studio Code

Imagen 2

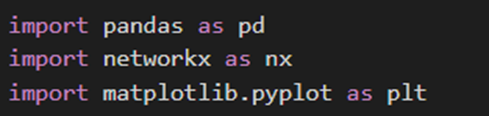


**Código**

Usamos el lenguaje Python para crear nuestro código, para iniciar comentaremos sobre la situación problema planteada por la docente a cargo.

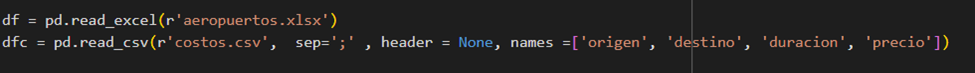
Dados dos archivos, uno llamado aeropuertos.xlsx y costos.csv, donde podemos ver en el primer archivo los aeropuertos en código IATA y en la segundo los vuelos disponibles, nosotros como grupo planteamos la situación problema he usamos las librerías de Python: pandas, networkx, matplotlib (Ver imagen 3)

Imagen 3



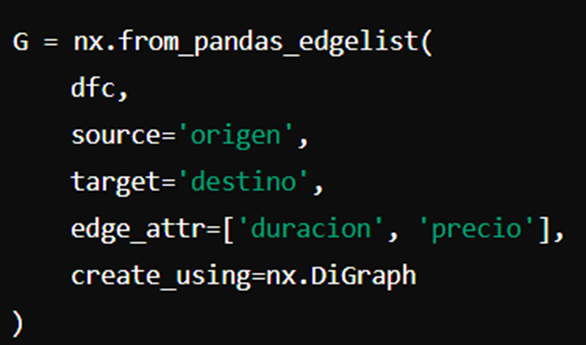
Luego utilizamos las siguientes dos variables con las extensiones de pandas para la lectura de datos (Ver imagen4)

Imagen 4



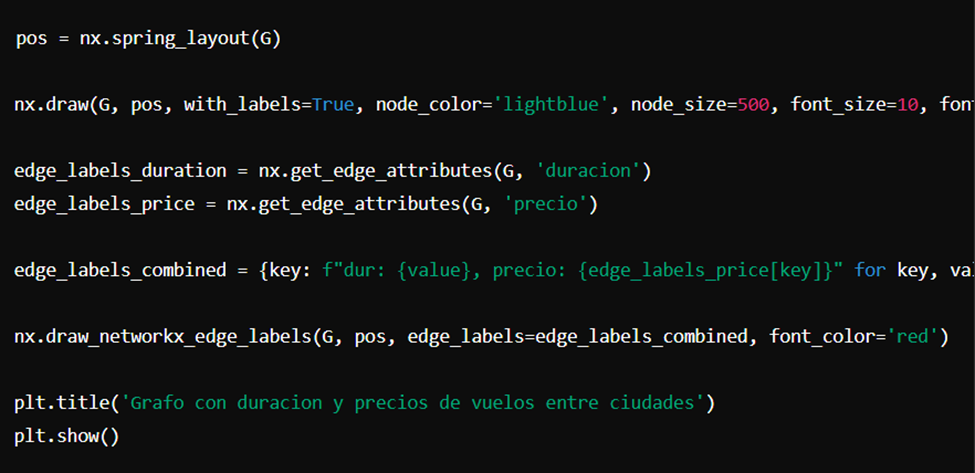
Utilizamos pandas y networkx para la creación de nuestro grafo (Ver imagen 5)

Imagen 5



En la siguiente parte del código, configuramos y dibujamos el grafo, configurando los nombres de los nodos y también dándole nombre a sus aristas con etiquetas (labels) (Ver imagen 6)

Imagen 6



**Funciones principales**

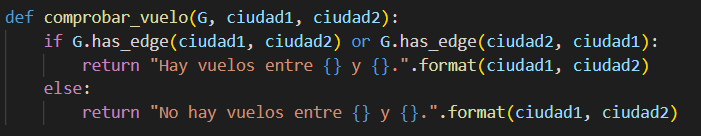
**Comprobar vuelo**

La función primero verifica si hay un borde de city1 a city2 o de city2 a city1 en el grafo G. Esto se hace usando el método has\_edge de G. El operador or se usa aquí para verificar ambas direcciones, lo que significa que la función devolverá verdadero si hay un vuelo en cualquier dirección entre las dos ciudades.

Si hay un borde (vuelo) entre las dos ciudades en cualquier dirección, la función devuelve una cadena que indica que hay vuelos entre city1 y city2. Esto se hace usando el método str.format de Python para insertar los nombres de las ciudades en la cadena.

Si no hay ningún borde (vuelo) entre las dos ciudades en ninguna dirección, la función devuelve una cadena que indica que no hay vuelos entre la ciudad 1 y la ciudad 2. Nuevamente, esto se hace utilizando el método str.format de Python para insertar los nombres de las ciudades en la cadena. (Ver imagen 7)

Imagen 7

****

**Ruta más barata**

La función de Python ruta\_mas\_barata calcula la ruta más barata entre dos ciudades en un grafo G. Cada borde tiene un peso asociado 'precio' que representa el costo de la ruta.

La función toma tres parámetros: G (el grafo), ciudad1 (la ciudad de partida) y ciudad2 (la ciudad de destino).

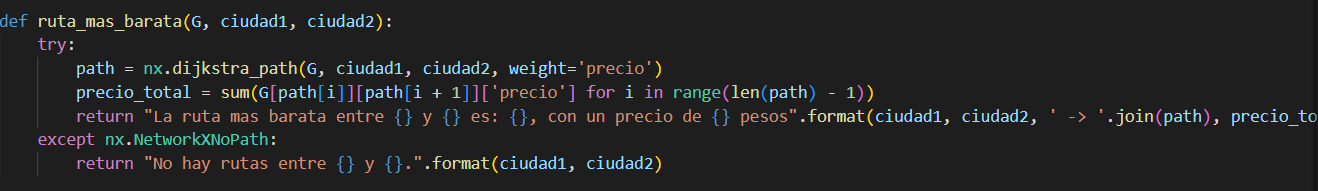
La función primero intenta encontrar la ruta más corta de ciudad1 a ciudad2 utilizando el algoritmo de Dijkstra, que es un algoritmo común para encontrar las rutas más cortas en un grafo. La función nx.dijkstra\_path se utiliza para este propósito, con 'precio' especificado como el parámetro de peso. Esto significa que el algoritmo considerará el atributo 'precio' de los bordes (es decir, el costo de las rutas) al determinar la ruta más corta.

Si se encuentra una ruta, la función calcula el costo total de esta ruta. Esto se hace iterando sobre la ruta (que es una lista de nodos/ciudades), accediendo al atributo 'precio' del borde entre cada par de ciudades consecutivas y sumando estos costos. La función sum incorporada de Python se utiliza para este propósito, junto con una expresión generadora que genera los costos.

Luego, la función devuelve una cadena que describe la ruta más barata y su costo total. El método str.format se utiliza para insertar la ciudad de partida, la ciudad de destino, la ruta (como una cadena donde las ciudades están unidas por ' -> ') y el costo total en esta cadena.

Si no se encuentra ninguna ruta entre ciudad1 y ciudad2 (lo que puede suceder si las ciudades no están conectadas en el gráfico), la función nx.dijkstra\_path lanza una excepción NetworkXNoPath. La función captura esta excepción y devuelve una cadena que indica que no hay rutas entre las dos ciudades.(Ver imagen 8)

Imagen 8

****

**Tiempo total de vuelo**

La función tiempo\_total\_de\_vuelo calcula el tiempo total de vuelo entre dos ciudades en un gráfico. El grafo G representa una red de ciudades con aristas que representan los vuelos entre ellas. El peso de cada arista es la duración del vuelo entre dos ciudades.

La función toma tres parámetros: G (el gráfico), ciudad1 (la ciudad de partida) y ciudad2 (la ciudad de destino).

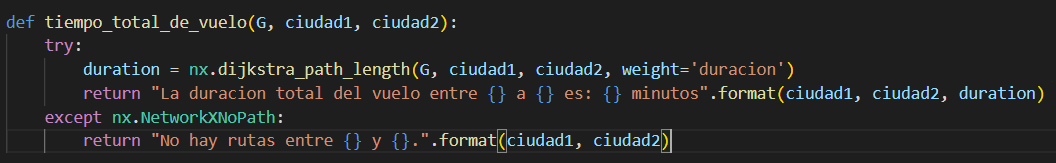
La función utiliza el algoritmo de Dijkstra para encontrar el camino más corto entre ciudad1 y ciudad2 basándose en el atributo 'duración' de las aristas. Esto se hace utilizando la función nx.dijkstra\_path\_length de la biblioteca NetworkX. El camino más corto en este contexto es el camino con la menor duración total.

El resultado de nx.dijkstra\_path\_length se almacena en la variable duración. Esto representa el tiempo total de vuelo entre ciudad1 y ciudad2.

La función luego devuelve una cadena que incluye los nombres de las ciudades de partida y destino y el tiempo total de vuelo.

Si no existe una ruta entre ciudad1 y ciudad2, nx.dijkstra\_path\_length genera una excepción nx.NetworkXNoPath. La función captura esta excepción y devuelve una cadena que indica que no existen rutas entre las dos ciudades.(Ver imagen 9)

Imagen 9

****

**Encontrar ruta**

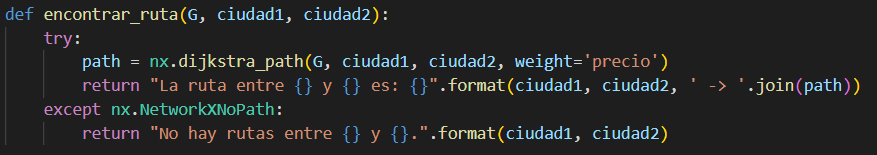
La función encontrar\_ruta(G, ciudad1, ciudad2) está diseñada para encontrar el camino más corto entre dos ciudades (ciudad1 y ciudad2) en un grafo G utilizando el algoritmo de Dijkstra. El peso de cada borde, que representa el costo de la ruta, se almacena en el atributo 'precio'.

La función comienza tratando de encontrar el camino más corto entre ciudad1 y ciudad2 utilizando la función nx.dijkstra\_path() de la biblioteca NetworkX. Esta función toma cuatro argumentos: el grafo G, el nodo de origen ciudad1, el nodo de destino ciudad2 y el atributo de peso del borde 'precio'. Devuelve una lista de nodos que forman el camino más corto de ciudad1 a ciudad2.

Si se encuentra un camino, la función formatea una cadena que describe el camino. El método join() se utiliza para concatenar los nodos en el camino en una cadena, con ' -> ' como separador. Esta cadena se inserta luego en la plantilla "La ruta entre {} y {} es: {}" utilizando el método format(), donde los marcadores de posición {} se reemplazan por ciudad1, ciudad2 y la cadena de ruta, respectivamente.

Si no se encuentra ninguna ruta, la función nx.dijkstra\_path() genera una excepción NetworkXNoPath. La función encontrar\_ruta() captura esta excepción y devuelve una cadena que indica que no hay rutas entre ciudad1 y ciudad2. Esta cadena se formatea utilizando el método format(), donde ciudad1 y ciudad2 reemplazan los marcadores de posición {} en la plantilla "No hay rutas entre {} y {}". (Ver imagen 10)

Imagen 10

****

**Ruta más rápida**

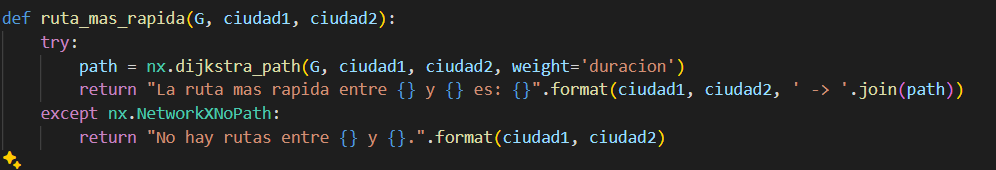
La función de Python ruta\_mas\_rapida está diseñada para encontrar la ruta más rápida entre dos ciudades en un grafo. La función toma tres parámetros: G, ciudad1 y ciudad2. G es un grafo NetworkX que representa la red de ciudades, donde cada nodo es una ciudad y cada borde representa una ruta entre dos ciudades. ciudad1 y ciudad2 son los nombres de las dos ciudades entre las que queremos encontrar la ruta más rápida.

La función utiliza el algoritmo de Dijkstra para encontrar el camino más corto entre ciudad1 y ciudad2. Este algoritmo se implementa mediante la función nx.dijkstra\_path de la biblioteca NetworkX. El parámetro de peso de esta función se establece en 'duración', lo que significa que el algoritmo considerará la duración de las rutas para determinar el camino más rápido.

Si se encuentra una ruta, la función devuelve una cadena que describe la ruta más rápida. El método join se utiliza para concatenar los nombres de las ciudades en la ruta, separados por ' -> '.

Si no existe ninguna ruta entre ciudad1 y ciudad2, la función nx.dijkstra\_path genera una excepción NetworkXNoPath. Esta excepción se captura en el bloque except y la función devuelve una cadena que indica que no existen rutas entre las dos ciudades. (Ver imagen 11)

Imagen 11

****

**Interacción con el usuario**

Una vez definidas las ciudades de origen y destino, el código entra en un bucle infinito, presentando al usuario un menú de opciones. Las opciones son:

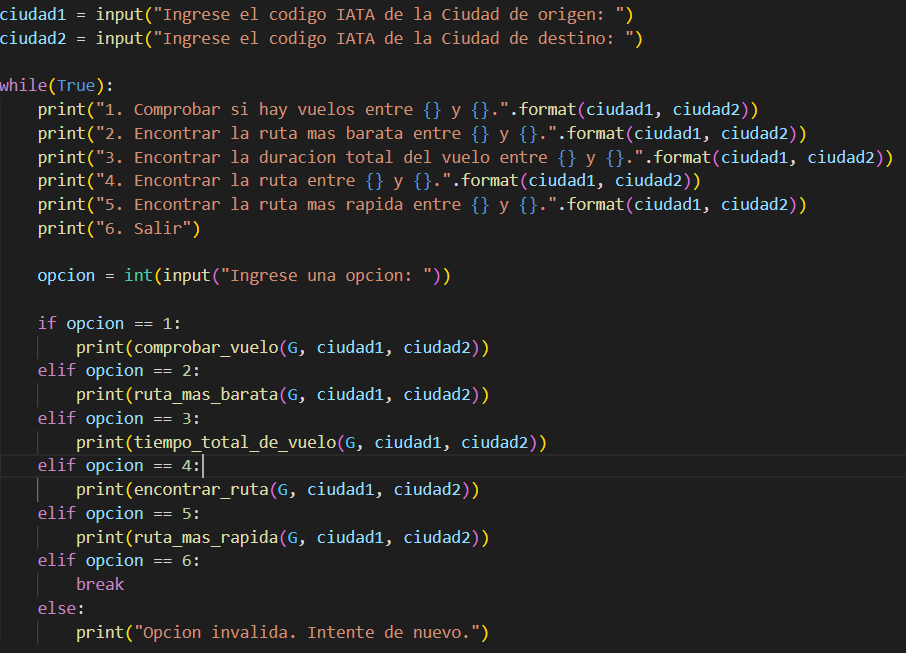
1. Verificar si hay vuelos entre las dos ciudades.
2. Encontrar la ruta más barata entre las dos ciudades.
3. Encontrar la duración total del vuelo entre las dos ciudades.
4. Encontrar la ruta entre las dos ciudades.
5. Encontrar la ruta más rápida entre las dos ciudades.
6. Salir del programa.

Se le solicita al usuario que ingrese un número correspondiente a una de estas opciones. La entrada se convierte a un entero utilizando la función int ().

Dependiendo de la elección del usuario, se llama a una función diferente. Cada función toma como argumentos un gráfico G, y las ciudades de origen y destino. Se espera que las funciones devuelven una cadena con la información solicitada, que luego se imprime en la consola.

Si el usuario ingresa '6', se ejecuta la sentencia break, que sale del bucle y finaliza el programa. Si el usuario ingresa un número que no corresponde a ninguna opción, se imprime un mensaje y se muestra nuevamente el menú. (Ver imagen 12)

Imagen 12

****