PROYECTO FINAL

INTEGRANTES:

Daniel Guarín, Sara Gómez López, Santiago Betancur

Estructura de datos y algoritmos

Gloria Stella Sepúlveda

Universidad EAFIT

Medellín

Descripción del proyecto:

El algoritmo permite crear un grafo completo que se genera a partir del archivo "**costos.csv**", también permite encontrar la ruta más cercana una vez se le ingrese la ciudad desde la que desea viajar el usuario. Además, se le da un tiquete en el avión más próximo de acuerdo a la ciudad de destino que desee incluyendo el precio de la ruta.

Documentación del código – Manual de usuario:

Librerías utilizadas:

- **networkx:** Es una librería de Python que nos permite trabajar dinámicamente las funcionalidades de los grafos. Esta proporciona clases y funciones para trabajar con grafos.
- **matplotlib.pyplot:** Es una librería de trazado 2D que proporciona una interfaz para crear gráficos y visualizaciones. Se utiliza en este proyecto para visualizar el grafo y las rutas de vuelo utilizando la función **'draw()'** y **'draw_networkx_edge_labels()'** para etiquetar las aristas con los precios.
- **pandas:** Utilizada para la manipulación y análisis de datos. En este proyecto, se utiliza para leer y escribir archivos CSV y para manipular los datos de rutas de vuelo en forma de DataFrame.
- **csv:** Se utiliza para leer y escribir archivos de tipo CSV (valores separados por comas). Se utiliza en este proyecto para escribir nuevas rutas de vuelo en el archivo CSV y para leer los datos de rutas de vuelo existentes.
- **datetime:** Se utiliza para trabajar con fechas y horas. En este proyecto, se utiliza para generar la hora de salida de los vuelos simulados en la clase '**Flight'**.

- **random:** Se usa para generar números aleatorios y realizar selecciones aleatorias. Se utiliza en este proyecto para seleccionar aerolíneas, generar números de vuelo y generar tiempos aleatorios para los vuelos simulados en la clase **'Flight'**.

Funcionalidades del proyecto:

- El constructor de la clase **'Graph'** nos sirve para leer el archivo CSV y crea el grafo a partir de los datos proporcionados. Además, convierte los precios de las rutas de los vuelos a un número entero para que sea fácil trabajar con los precios, ya que inicialmente eran valores de tipo **String.**
- **showGraph():** Muestra el grafo utilizando la librería **Matplotlib**. Los nodos se representan como círculos etiquetados con el nombre del aeropuerto y las aristas se representan como fechas dirigidas con la etiqueta de los precios.
- **addNewRoute(origin, destination, price):** Agrega una nueva ruta de vuelo al grafo. Toma como argumentos el aeropuerto de origen, el aeropuerto de destino y el precio de la ruta. La nueva ruta se guarda en el archivo CSV y se vuelve a inicializar el constructor. De esa forma, se actualiza el grafo completo y se reflejan los cambios.
- **shortestRoute(origin, destination):** Encuentra la ruta más corta entre dos aeropuertos utilizando el algoritmo de la ruta más corta de Dijkstra. Muestra el grafo completo con la ruta más corta resaltada en rojo. También pregunta al usuario si desea ver los boletos de los vuelos individuales en la ruta. Si el usuario desea verlos, se genera un archivo CSV en el cuál se van a mostrar vuelos próximos a despegar con la ruta dada por el usuario.
Una vez generado el archivo, al usuario se le genera un ticket en el avión que esté más próximo a salir y en caso de que no se genere ningún avión, al usuario se le mostrará un mensaje diciendo que no hay vuelos disponibles para esa ruta.

- **files_generator(ori, dest):** Genera archivos CSV simulando vuelos entre los aeropuertos de origen y destino dados. Cada archivo contiene información aleatoria sobre los vuelos, como la aerolínea, el número de vuelo, la hora de salida, etc. Esta función se utiliza para simular la disponibilidad de vuelos en el sistema.

Ese fue un resumen de las funcionalidades que tienen las funciones más importantes del código.

Por otro lado, contamos con la clase **Flight** que representa un vuelo y todo lo que caracteriza a uno como:

- Aerolínea.
- Número de vuelo.
- Aeropuerto de origen y destino.
- Hora de salida.
- Duración.
- Prioridad.

Descripción de la clase DataFrame:

- En el constructor se inicializa la clase y se carga un archivo de Excel llamado **“aeropuertos.xlsx”** en un dataframe usando **pd.read_excel()**. Luego, se eliminan los duplicados del dataframe usando **drop_duplicates()**.

El dataframe resultante se le asigna a la variable de instancia **self.df**.

- **showDf()** imprime el dataframe almacenado en la variable de instancia **self.df**.
- **searchCity()** recibe un argumento **"city"** y busca en el dataframe el nombre de una ciudad coincidente. Recorre las filas del dataframe y compara el valor de la columna 2 con el nombre de la ciudad proporcionada. Si encuentra una coincidencia, devuelve el valor de la columna 0

correspondiente a ese registro. Si no encuentra ninguna coincidencia, imprime **"Invalid name"** y devuelve **False**.

- **getAirports()** recorre el dataframe y extrae los valores de la columna 2, eliminando el carácter especial "\xa0" en cada valor. Luego, devuelve una lista con todos los nombres de aeropuertos.
- **getIATA()** recorre el dataframe y extrae los valores de la columna 0, que representa los códigos IATA de los aeropuertos. Luego, devuelve una lista con todos los códigos IATA.
- **searchAirportName()** recibe un argumento **"IATA"** y busca en el dataframe el nombre de un aeropuerto correspondiente al código IATA proporcionado. Recorre las filas del dataframe y compara el valor de la columna 0 con el código IATA. Si encuentra una coincidencia, devuelve el valor de la columna 1 correspondiente a ese registro.
- **iatawithname()** recibe un argumento **"name"** y busca en el dataframe el código IATA correspondiente a un nombre de aeropuerto proporcionado. Recorre las filas del dataframe y compara el valor de la columna 1 con el nombre del aeropuerto. Si encuentra una coincidencia, devuelve el valor de la columna 0 correspondiente a ese registro.
- **searchIATA()** recibe un argumento **"IATA"** y busca en el dataframe el nombre de un aeropuerto correspondiente al código IATA proporcionado. Recorre las filas del dataframe y compara el valor de la columna 0 con el código IATA. Si encuentra una coincidencia, devuelve el valor de la columna 2 correspondiente a ese registro.

Descripción del menú:

- El método de inicialización crea una instancia de la clase Graph y la asigna a la variable **self.database**. Esto permite acceder a los métodos de la clase Graph desde la instancia de Menu.
- **mainMenu(self):** Este método muestra el menú principal y gestiona las opciones seleccionadas por el usuario. Utiliza el método **showMenu()** para obtener la opción

seleccionada por el usuario y luego ejecuta la acción correspondiente según la opción seleccionada.

- **showMenu():** Este método muestra el menú de opciones al usuario y lee la opción seleccionada. Retorna la opción como un entero para su procesamiento posterior.
- **addNewRoute(self):** Este método permite al usuario agregar una nueva ruta ingresando el aeropuerto de origen, el aeropuerto de destino y el precio de la ruta. Lee los valores ingresados por el usuario y luego llama al método **addNewRoute()** de la instancia de Graph para agregar la nueva ruta.
- **mostEconomic(self):** Este método permite al usuario encontrar la ruta más económica entre aeropuertos. Primero muestra al usuario los aeropuertos disponibles como puntos de partida (origen) y le pide que ingrese el aeropuerto de origen. Luego, utiliza el método **findNeighbors()** para encontrar los aeropuertos vecinos (destinos) a partir del aeropuerto de origen ingresado. Muestra al usuario los destinos disponibles y le pide que ingrese el aeropuerto de destino. Finalmente, llama al método **shortestRoute()** de la instancia de Graph para encontrar y mostrar la ruta más económica entre el aeropuerto de origen y destino.

Finalmente, en la clase **Main**, importamos la clase **Menu** y llamamos al método **mainMenu()** para inicializar el programa y mostrar el menú de opciones.