UNIDAD TEMÁTICA 4 – GRAFOS DIRIGIDOS– Trabajo de Aplicación 1

ESCENARIO

Una compañía de aviación nacional necesita representar los vuelos que opera en diferentes aeropuertos y calcular cuáles son los viajes de menor costo que permiten unir esos aeropuertos.

Los aeropuertos en los que opera son:

> Artigas, Canelones, Colonia, Durazno, Florida, Montevideo, Punta del Este y Rocha.

Los vuelos operados son (origen, destino, distancia):

> Artigas, Rocha, 400; Canelones, Artigas, 500; Canelones, Colonia, 200; Canelones, Durazno, 170; Canelones, Punta del Este, 90; Colonia, Montevideo, 180; Florida, Durazno, 60; Montevideo, Artigas, 700; Montevideo, Canelones, 30; Montevideo, Punta del Este, 130; Punta del Este, Rocha, 90; Rocha, Montevideo, 270

El analista a quien se le ha encargado la tarea ha decidido que la mejor forma de solucionar el problema es mediante el uso de grafos dirigidos.

EJERCICIO 1

1) Expresar el grafo como:
   a) Matriz de adyacencias
   b) Lista de adyacencias
2) Representarlo gráficamente.

EJERCICIO 2

Aplicando paso a paso el método de Dijkstra, calcular el vuelo de menor costo (distancia) entre el aeropuerto de Montevideo y cada uno los otros aeropuertos. ¿Cuál es el trayecto recorrido? (salida – escalas – destino)

EJERCICIO 3

Ahora que hemos introducido el TDA Grafo Dirigido, y empezamos a conocer las operaciones que el mismo debería soportar, deseamos analizar diferentes formas de diseñar su estructura a efectos de lograr la mayor eficiencia en las operaciones y consumo de memoria razonable.

A la vez, nos interesa para esta tarea hacer uso avanzado de las **colecciones y clases disponibles en la librería** del lenguaje, así como de **genéricos**.

Se requiere:

- Diseñar a alto nivel la arquitectura del TDA Grafo, indicando las operaciones a implementar, las estructuras a usar y el orden del tiempo de ejecución correspondiente. **Dibujar las estructuras (clases) y relaciones entre las mismas.**
- Evaluar la aplicabilidad de diferentes colecciones disponibles en la API de colecciones de JAVA para lograr un diseño más eficiente y compacto (es decir, a la vista de las operaciones que son requeridas, comparar la implementación usando diferentes combinaciones de colecciones disponibles en la API de JAVA – a la luz del consumo de memoria y tiempo de ejecución)