TPO programación

INTEGRANTES DEL GRUPO:
Bergamo Thomás (LU 1183936) - Corti Leandro (LU 1184844) - Favereau Dante (LU 1190207) - Minian Thiago (LU 1187596).

Profesor:
Pérez Nicolás Ignacio.

Cursada:
Turno mañana, día viernes (Ingeniería en informática).

Objetivos del proyecto:

Este trabajo tuvo como objetivo implementar de forma práctica algún algoritmo de caminos mínimos en grafos ponderados utilizando java, para resolver una problemática en cuestión. A su vez, se buscó reforzar conceptos como:

- Uso de gafos con estructuras dinámicas.
- Programación orientada a objetos.
- Uso de interfaces e implementación de estructuras genéricas.
- Organización modular del código y buenas prácticas de diseño.

Estructura del código:

El proyecto fue dividido en clases e interfaces con responsabilidades bien definidas, establecidas con la clase <T> (genérica) para poder reutilizar el código en caso que se lo necesite:

- **IGrafo<T>:** Interfaz que define las operaciones principales del grafo, agregar nodos, agregar aristas, obtener caminos mínimos, etc.

- **INodo<T>:** Interfaz que representa un nodo dentro del grafo, con acceso a sus vecinos y su valor.

- **Grafo<T>:** Clase principal que implementa la estructura del grafo utilizando mapas y listas. Administra los nodos y aplica el algoritmo de Djikstra.

- **Nodo<T>:** Clase concreta que implemente la interfaz INodo. Representa una ciudad y mantiene sus vecinos con los respectivos pesos.

- **GrafoUtils:** Clase auxiliar para mostrar resultados por pantalla, validar caminos y presentar ciertas características sobre el resultado.

Esta estructura modular permitió desacoplar la lógica del grafo y facilitar futuras ampliaciones, como cambiar de algoritmo o adaptar el sistema a otros dominios.

Implementación:

Para esto, creamos un programa en el cual los usuarios que lo utilicen ingresan los nombres de las ciudades que conforman el país elegido y la distancia en kilómetros entre cada una de ellas (pueden incluir todas o solamente las que consideren necesarias para que el programa funcione). A medida que se van ingresando las ciudades, el programa las transforma en nodos, cada uno con su nombre e ID (luego utilizado para el algoritmo dijkstra) y va ingresando esos nodos en un grafo que conecta todas las ciudades entre sí, mientras las aristas simulan, de esta manera, la distancia que hay entre las mismas.

Una vez terminado el grafo, cuando se quiere saber el recorrido mínimo entre dos ciudades, se llama a la función que implementa dijkstra. Lo que hace es analizar todos los caminos posibles para llegar del origen al destino calculando los costos (cantidad de kilómetros recorridos) y descartando aquellos caminos que requieren más costo del que se obtuvo yendo por otra parte. De esta manera, cuando el programa finaliza muestra mediante un listado como se llega del origen al destino de la manera más corta posible, junto con la distancia total recorrida.

Una gran ventaja de este algoritmo es que solamente es necesario ejecutarlo una vez, ya que, si no se cambian las rutas o se agregan nuevas ciudades de por medio, el camino más corto va a seguir siendo siempre el mismo

Ejemplo de uso:

Se ingresaron las siguientes ciudades: Buenos Aires, Córdoba, Rosario, Mendoza y Salta. Luego, se agregaron aristas ponderadas que representan las distancias entre dichas ciudades.

A continuación, se solicitó el camino más corto entre Buenos Aires y Salta. La salida del programa fue:

Camino más corto:
Buenos Aires -> Córdoba -> Salta
Distancia total: 1278km

Este resultado demuestra el correcto funcionamiento del algoritmo, el cual recorrió los caminos intermedios y devolvió la ruta óptima.

¿Por qué Djikstra?

El algoritmo de Djikstra fue elegido por su eficiencia y simplicidad para grafos ponderados sin pesos negativos. Además, permite obtener resultados óptimos de forma rápida cuando se busca la ruta más corta desde un punto de origen específico.

Ventajas:

- Encuentra el camino más corto desde un único origen a todos los destinos.
- Utiliza estructuras eficientes como colas de prioridad.
- Es fácil de implementar y comprender.

Desventajas:

- No puede trabajar con aristas de peso negativo.
- Su rendimiento disminuye en grafos muy densos sin optimizaciones adicionales.

Comparación entre Djikstra y Floyd-Warshall:

Durante la cursada también analizamos distintos algoritmos que permite resolver problemas de caminos mínimos en grafos, como por ejemplo Floyd-Warshall. Cada uno tiene sus particularidades y aplicaciones específicas dependiendo del tipo de grafo, el objetivo del problema y los requerimientos de eficiencia.

Djikstra es más eficiente en estos casos puntuales, ya que calcula la distancia mínima desde un único nodo origen hacia todos los demás, lo que lo hace ideal cuando solo se necesitan algunas rutas. Su complejidad es menor que Floyd-Warshall, especialmente en grafos escasamente conectados.

Por otro lado, **Floyd-Warshall** resuelve el problema de caminos más cortos entre todos los pares de nodos, lo que implica una mayor carga computacional. Este enfoque sería útil si se necesitara consultar frecuentemente entre muchas combinaciones de ciudades, pero no resulta necesario ni eficiente para el caso planteado en este trabajo.

En conclusión, **Djikstra** fue la elección más conveniente tanto por rendimiento como por simplicidad, ajustándose al alcance y necesidades del sistema.

Aplicación práctica: logística de una empresa de envíos:

Una posible aplicación real del programa desarrollado es en una empresa de logística o reparto, que necesita optimizar sus rutas de entrega entre distintas ciudades del país.

Por ejemplo, una empresa de envíos llamada "ExpressTransport" debe entregar paquetes desde su central en Buenos Aires hacia sucursales en otras provincias. Como el transporte tiene un costo asociado a la distancia, buscan minimizar ese costo utilizando el camino más corto entre ciudades.

Gracias a este sistema, cargan una vez el mapa con las conexiones y distancias entre las ciudades, y luego el algoritmo de Dijkstra les permite obtener rápidamente la ruta óptima para cada envío.

Esto no solo reduce costos de combustible y tiempo, sino que además permite responder más rápido a los clientes con estimaciones precisas de entrega.

Conclusión:

A través de este trabajo práctico pudimos reforzar el conocimiento sobre algoritmos de grafos y su implementación en java utilizando buenas prácticas de programación. Además, mejoramos el trabajar en equipo, organizar nuestras tareas y aplicar estructuras abstractas como interfaces y genéricos para crear un sistema flexible, reutilizable y fácil de mantener.