Trabajo Práctico Obligatorio Programación II

1. Problema

En muchas ciudades modernas, uno de los desafíos logísticos más importantes es la **optimización de las rutas de recolección de residuos**. Una mala planificación genera recorridos innecesarios, mayor consumo de combustible, desgaste de vehículos, y sobre todo, una mala prestación del servicio.

Este trabajo plantea una solución a ese problema utilizando conceptos de teoría de grafos. El objetivo es representar los distintos **barrios como nodos** y las **calles como aristas**, donde el peso representa el **tiempo estimado de tránsito**.

La meta principal es hallar las rutas más cortas desde el **centro de operaciones** (origen) hacia todos los barrios, minimizando el tiempo total de recorrido de los camiones recolectores.

Este tipo de modelado es habitual en logística, transporte urbano y planificación urbana, y se presta a soluciones algorítmicas eficientes.

2. Solución utilizando el algoritmo de Dijkstra

Para este problema se utilizó el **algoritmo de Dijkstra**, que es ampliamente conocido por resolver eficientemente el **problema del camino más corto desde un único nodo origen** hacia todos los demás en un grafo dirigido con **pesos no negativos**.

¿Por qué Dijkstra?

- Es **rápido y eficiente** con grafos densos o moderadamente grandes.
- Tiene **complejidad O ((V + E) log V)** usando una cola de prioridad. V: la cantidad de nodos (barrios). E: la cantidad de aristas (calles).
- No necesita visitar nodos ya optimizados.
- Es ideal para problemas donde se quiere ir **desde un punto fijo hacia todos los demás**.

Dijkstra comienza desde el nodo origen y **propaga las distancias mínimas** hacia todos los vecinos, actualizando cada vez que se encuentra una ruta más corta. Se apoya en una cola de prioridad para siempre trabajar con el nodo de menor distancia estimada.

3. Implementación utilizando TDA (Tipo de Dato Abstracto)

La solución fue desarrollada en el lenguaje de programación **Java**, aplicando buenas prácticas de diseño orientado a objetos y **modularización mediante TDAs**.

Estructura del proyecto:

- **Interfaces:**
  - IGrafo: métodos para construir y recorrer el grafo.
  - INodo: nodo con barrio, distancia mínima y estado de visita.
  - IBarrio: representa cada barrio con su nombre.
- **Modelos:**
  - Barrio: implementación de IBarrio.
  - Nodo: implementación de INodo.
  - Grafo: implementación completa de IGrafo, con lista de adyacencia, Dijkstra y matriz de adyacencia.
- **Test (clase principal):**
  Se instancian 5 barrios y se agregan calles entre ellos con tiempos variados. Luego se ejecuta el algoritmo de Dijkstra desde el centro y se imprime:
  - El **camino más corto desde el centro** a cada barrio.
  - Una **matriz de adyacencia** clara y alineada con valores de tiempos o "∞" cuando no hay conexión directa.

<u>Fragmento de salida esperada:</u>

Desde Centro hasta Barrio Norte: 5 minutos
Desde Centro hasta Barrio Este: 8 minutos
Desde Centro hasta Barrio Oeste: 10 minutos
Desde Centro hasta Barrio Sur: 7 minutos

Matriz de Adyacencia:

	Barrio Este	Barrio Norte	Barrio Oeste	Barrio Sur	Centro
Centro	∞	2	∞	∞	∞
Barrio Este	3	∞	2	∞	∞
Barrio Norte	∞	∞	∞	∞	6
Barrio Oeste	∞	∞	4	∞	∞
Barrio Sur	∞	5	∞	7	∞

4. Diferencias con otros algoritmos vistos en clase

Durante la cursada se abordaron también los algoritmos de **Prim**, **Kruskal** y **Bellman-Ford**. A continuación se detalla una comparación con Dijkstra:

Algoritmo	**Uso principal**	**Pesos negativos**	**Caso ideal**
Dijkstra	Camino más corto desde un único nodo	✗ No	Mapas urbanos, GPS, logística
Prim	Árbol de expansión mínima (todos los nodos)	✗ No	Redes eléctricas o de agua
Kruskal	Árbol de expansión mínima (ordenado por peso)	✗ No	Conexión mínima entre puntos aislados
Bellman-Ford	Caminos mínimos con pesos negativos	✓ Sí	Finanzas, redes con penalizaciones

Dijkstra fue seleccionado porque:

- Es el **más adecuado para grafos sin pesos negativos**.
- Resuelve específicamente el tipo de problema planteado: **rutas desde un único origen**.
- Es más eficiente que Bellman-Ford para este escenario.

5. Conclusión

Este trabajo práctico demuestra cómo una situación real puede modelarse eficazmente mediante teoría de grafos y estructuras abstractas de datos.

La implementación en Java con TDA permite una arquitectura **modular y escalable**, adaptable a ciudades más grandes o problemas similares como:

- **Distribución de productos**.
- **Diseño de redes de transporte**.
- **Planificación de rutas escolares**.
- **Tareas de patrullaje urbano**.

El uso de una **matriz de adyacencia visual**, además del algoritmo de Dijkstra, facilita tanto la comprensión del problema como su resolución computacional.