Trabajo Práctico Obligatorio

Tema: Ruta más corta en una red de colectivos

Alumnos: Ignacio Furfaro, Bautista Failo, Julian Conti

1 . Problema planteado:

Se seleccionó como caso de estudio la representación de una **red de colectivos urbanos**. Cada nodo del grafo representa una **parada de colectivo**, y cada arista representa un **tramo entre dos paradas**, ponderado según la duración estimada del viaje (en minutos).

El objetivo fue implementar un sistema que permita encontrar la ruta más corta (en tiempo) entre dos paradas cualquiera utilizando el algoritmo de Dijkstra.

2. Solución Propuesta

- La estructura del grafo ponderado fue desarrollada mediante un TDA llamado `GrafoPonderado<T>`, parametrizado con tipos genéricos. Este grafo almacena los vértices y sus conexiones mediante un `Map<T, Map<T, Integer>>`, donde la clave es el nodo de origen y el valor es un mapa de destinos con sus respectivos pesos.
- La implementación no utiliza clases `Nodo` ni `Arista`, permitiendo una estructura más liviana y directa. El algoritmo de Dijkstra fue implementado para calcular tanto el camino más corto (`dijkstra(T origen, T destino)`) como la distancia mínima a todos los nodos (`dijkstraDistancias(T origen)`)

3. Cómo se implementó:

- El grafo es dirigido y ponderado, usando `Map<T, Map<T, Integer>>` para representar conexiones.
- Dijkstra calcula dos cosas:
    - El **camino exacto** entre dos paradas (`List<T> dijkstra(origen, destino)`).
    - Las **distancias mínimas** desde una parada a todas (`Map<T, Integer> dijkstraDistancias(origen)`).
- Usamos una `PriorityQueue` para siempre procesar el nodo más cercano.

Ejemplo:

Si conectamos:

- Plaza → Facultad (5 min)
- Plaza → Terminal (10 min)
- Facultad → Terminal (3 min)

Dijkstra desde *Plaza* detecta que el camino más corto a Terminal es:

`Plaza → Facultad → Terminal (total: 8 min)`

en lugar de Plaza → Terminal directo (10 min).

4. Complejidad:

- Tiempo: `O(V^2)` para implementación simple, o `O((V + E) log V)` con `PriorityQueue` y estructuras eficientes (como hicimos).
- Espacio: `O(V)` para distancias, predecesores y visitados.

5. Limitaciones:

- No funciona con pesos negativos.
- Solo sirve para una **fuente fija**, no para todos los pares (ahí conviene Floyd).

6. Comparación con otros Algoritmos

Floyd-Warshall:

- Calcula la **distancia mínima entre todos los pares** de nodos.
- Más costoso computacionalmente (`O(V^3)`).
- Útil si queremos tener la matriz completa de distancias.

Kruskal:

- Encuentra un Árbol de Expansión Mínima (MST).

No se usa para caminos entre nodos específicos, sino para **conectar todos los nodos con el menor costo total**.

No aplicable directamente a nuestro caso (no se busca un MST).

	Algoritmo visto en clase	**Algoritmo del TPO**
Tipo de nodos	Usa `Nodo` con ID entero (`int`)	Usa `Vertice<T>` genérico (puede `ser `String`, `Persona`, etc.)
Identificación	Por `int` (por ejemplo, nodo 1, 2, 3...)	Por objeto genérico (`"ParadaA"`, `"ParadaB"`, etc.)
Interfaz	No hay interfaces (`INodo` se menciona pero no se respeta en todo el código).	Usa interfaces definidas (`IGrafo`, `IVertice`) que estructuran y encapsulan funciones.
Acoplamiento	Código muy acoplado y todo embebido en la clase `Dijkstra.`	Separación clara en paquetes `modelo`, `interfaces`, y `test` con uso de TDA.
Modularidad	Clase interna para `NodoDistancia` y lógica todo en un método.	Algoritmos separados, reutilizables y adaptables a cualquier tipo de dato.

7. Pruebas y Resultados

Se crearon varios tests con escenarios de ejemplo:

- Red de 5-6 paradas conectadas.

- Prueba de caminos directos y caminos con desvíos más cortos.
- Verificación de matriz de conexiones.

Los resultados confirman que el algoritmo encuentra correctamente el camino óptimo, y que se puede reutilizar fácilmente para diferentes casos.

Al ejecutar el testGrafoPonderadoMain : Obtenemos esta respuesta