Problema de Caminos Mínimos – Version 2

1. Descripción del problema

Una empresa de transporte necesita planificar rutas óptimas desde su depósito hacia varias tiendas.

Objetivos principales:

- Minimizar el coste de transporte (distancia o tiempo).
- Obtener caminos optimizados para planificar rutas de manera eficiente.

Se modela mediante un **grafo ponderado**, donde:

- Los nodos representan ubicaciones (depósito, tiendas).
- Las aristas representan rutas con un peso asociado (distancia o tiempo).

2. Representación con TDAs en Java

Para implementar el problema se utilizaron los siguientes TDAs:

TDA	Descripción
`Nodo<T>`	Representa un nodo genérico (depósito, tienda).
`Arista<T>`	Representa una arista con peso entre dos nodos.
`GrafoPesado<T>`	Grafo ponderado con lista de nodos y matriz de adyacencia de pesos.
Método `dijkstra(int origen)`	Calcula los caminos mínimos desde un nodo origen a todos los demás.

Ejemplo de creación del grafo en Java:

```
GrafoPesado<Punto> grafo = new GrafoPesado<>(4);
grafo.agregarNodo(new Punto("Depósito"));
grafo.agregarNodo(new Punto("Tienda A"));
grafo.agregarNodo(new Punto("Tienda B"));
grafo.agregarNodo(new Punto("Tienda C"));

grafo.agregarArista(0, 1, 10);
grafo.agregarArista(0, 2, 15);
grafo.agregarArista(1, 3, 12);
grafo.agregarArista(2, 3, 10);

int[] dist = grafo.dijkstra(0);
grafo.mostrarDistancias(dist);
```

3. Algoritmo Dijkstra

Dijkstra calcula los caminos de menor costo desde un nodo origen a todos los demás.

Procedimiento:

1. Inicializar todos los nodos con distancia infinita.
2. Seleccionar el nodo origen y asignarle distancia 0.
3. Considerar el nodo seleccionado como actual.
4. Marcar el nodo como visitado y actualizar las distancias de sus vecinos.
5. Repetir hasta visitar todos los nodos.

El resultado es un **arreglo de distancias mínimas** desde el nodo origen a cada nodo del grafo.

4. Comparación con otros algoritmos

Algoritmo	**Propósito**

Dijkstra	Camino mínimo desde un nodo origen en grafos ponderados sin aristas negativas.
Prim	Árbol de expansión mínima que conecta todos los nodos con menor costo total.
Kruskal	Árbol mínimo evitando ciclos, uniendo todos los nodos con menor costo.

Diferencia clave:
Dijkstra encuentra caminos mínimos desde un origen específico, mientras que Prim y Kruskal construyen árboles de costo mínimo para todo el grafo.

5. Ejemplo de ejecución

Se construyó un grafo ponderado con 4 nodos:

Nodo	**Índice**
Depósito	0
Tienda A	1
Tienda B	2
Tienda C	3

Aristas con peso:

Desde	**Hacia**	**Peso**
0	1	10
0	2	15
1	3	12
2	3	10

Salida de `TestDijkstra.java`:

```
Distancias desde Depósito:
Depósito: 0
Tienda A: 10
Tienda B: 15
Tienda C: 22
```

6. Grafo dirigido

Para que el grafo sea **dirigido**, se utiliza el parámetro `dirigido = true` al crear el grafo:

```
Grafo<Persona> grafo = new Grafo<>(5, true);
```

- Solo se agregan aristas en la dirección "desde → hacia".
- **DFS y BFS funcionan igual**, siguiendo la dirección de las aristas:
    - Solo se visitarán nodos accesibles a través de aristas salientes.
    - Por ejemplo, si existe `0 → 1` pero no `1 → 0`, desde el nodo 1 **no se llega al nodo 0**.

7. Comentarios y conclusiones

- La implementación utiliza **TDAs genéricos** (`Nodo<T>`, `Arista<T>`, `GrafoPesado<T>`), lo que permite reutilizar el grafo para distintos tipos de nodos (como `Punto` o `Persona`).
- El algoritmo Dijkstra es eficiente para **planificación de rutas** y cálculo de caminos mínimos en grafos ponderados.
- Limitación: solo grafos **sin aristas negativas**.
- Se cumplen los requisitos de la consigna: explicación del problema, solución, testeo bajo TDA y comparación con otros algoritmos.