# Trabajo Práctico Obligatorio
# Algoritmos y Estructuras de Datos
# 2

Alumnos:
-Bellinzona, Mateo
-Bonomo, Tobias
-Vargas, Facundo Gabriel

Profesor: Perez, Nicolas Ignacio

Consigna:
Buscar un problema real que se resuelva con alguno de los 3 algoritmos de Grafos Pesados que vimos en clase.
Crear o conseguir un programa que muestre este problema resuelto con alguno de estos algoritmos, bajo TDA.
En el repositorio del Proyecto crear un PDF breve (3/4 paginas) donde se explique el problema, la solución y las diferencias si las hay, con el algoritmo mostrado en clase.
Además si en la investigación apareció algún otro algoritmo comentarlo y buscar un testeo bajo TDA.

1. Descripción del problema real:
El trabajo práctico se basa en la optimización de rutas dentro de una red logística que conecta un centro de distribución con diferentes sucursales y clientes. El objetivo principal es minimizar el tiempo de entrega desde el centro logístico hacia cada destino, considerando los diferentes tiempos de traslado entre los puntos. Este tipo de optimización es clave en el ámbito logístico para mejorar la eficiencia operativa y reducir demoras en la entrega de paquetes.

2. Explicación de la solución:
Para resolver el problema, se modeló la red logística como un grafo dirigido y ponderado, donde:

-Los nodos representan los puntos de entrega (centro, sucursales, clientes).
-Las aristas representan las rutas posibles entre estos puntos.
-Los pesos de las aristas representan el tiempo estimado de viaje.

Por nuestra parte se decidió elegir e implementar el algoritmo de Dijkstra, ya que permite calcular de manera eficiente el camino más corto en términos de tiempo desde un nodo origen (el centro logístico) hacia todos los demás puntos del grafo.
A su vez, este algoritmo utiliza una cola de prioridad para seleccionar los nodos con menor tiempo acumulado e ir actualizando las distancias mínimas de forma progresiva. Esto garantiza que se obtenga el camino más rápido hacia cada destino.

Respecto al código, vamos a contar con las interfaces de Nodo y Grafo, pasando a sus implementaciones:

-Nodo:
Atributos:

```
private int id;
private String nombre;
private Map<InterfazNodo, Integer> vecinos;
```

Id: Identificador único del nodo.
Nombre

Vecinos: Representa los nodos vecinos conectados a este nodo.

Métodos de Nodo:
agregarVecino: Añade un nodo a la lista de vecinos del nodo actual, asociando un peso que representa el costo.

eliminarVecino: Elimina un nodo de la lista de vecinos, eliminando así la conexión entre ambos nodos.

getId(): Devuelve el identificador único del nodo.

-Grafo:
Atributos:

```
private Map<Integer, InterfazNodo> nodos;
```

Nodos: Almacena todos los nodos del grafo, utilizando su identificador como clave para acceder rápidamente a cada nodo.

Métodos de Grafo:
agregarNodo: Agrega un nuevo nodo al grafo si aún no existe uno con ese identificador. Crea una instancia de Nodo con el ID y nombre brindados.

eliminarNodo: Elimina un nodo del grafo y elimina todas las conexiones (aristas) que otros nodos pudieran tener hacia él.

agregarArista: Crea una conexión desde el nodo con id1 hacia el nodo con id2, asignándole un peso.

eliminarArista: Elimina la arista que conecta al nodo id1 con el nodo id2, “rompiendo” la relación entre esos dos nodos.

mostrarGrafo: Recorre todos los nodos del grafo e imprime sus vecinos con el peso de cada conexión.

obtenerPesoArista: Devuelve el peso de la arista que va desde el nodo id1 al nodo id2. Si no existe dicha conexión, devuelve -1.

3. Algoritmo alternativo y prueba con TDA:
Durante la investigación se consideró el algoritmo de Bellman-Ford como una posible alternativa para resolver caminos mínimos, ya que permite trabajar con pesos negativos. Sin embargo, se descartó su implementación en este trabajo debido a que el escenario del problema no contempla rutas con tiempos negativos, y su uso no resultaba necesario ni adecuado para este caso.

A modo de ejemplo, una prueba con TDA basada en Bellman-Ford podría verse así:

```
Map<Integer, Integer> distancias = AlgoritmoBellmanFord.bellmanFord(grafo, origen);
for (Integer destino : distancias.keySet()) {
    System.out.println("Destino " + grafo.getNombreNodo(destino) + ": " +
distancias.get(destino) + " minutos");
}
```

4. Conclusión:

El algoritmo de Dijkstra nos resultó una elección adecuada para este trabajo práctico, ya que permite encontrar rutas óptimas en términos de tiempo en un contexto de pesos positivos. Su implementación modular facilita la extensión del sistema a futuros escenarios logísticos más complejos, y permite adaptar el modelo a diferentes tipos de redes. Para casos con restricciones adicionales o escenarios con pesos negativos, podría considerarse en el futuro la incorporación de algoritmos como Bellman-Ford.