**Proyecto: Dijkstra con TDA**

1. Introduccion al problema

Este proyecto aplica el algoritmo de Dijkstra para resolver un problema real de caminos minimos en una ciudad.

El objetivo es encontrar el tiempo minimo de viaje desde un punto de origen hasta un destino, representando las calles como aristas y las intersecciones como vertices.

Se modela un mapa urbano donde cada zona (como Recoleta, Caballito o Palermo) es un vertice. Las conexiones entre ellas, con tiempos en minutos, son aristas con peso.

1. Estructura del grafo con TDA

La estructura se basa en listas implementadas como TDA, respetando las clases vistas en clase: Nodo y

Lista. Cada vertice contiene una Lista de Aristas que representa sus conexiones.

El grafo mantiene una Lista de Vertices. Cada Vertice contiene su nombre (String) y una Lista de Aristas, donde cada Arista tiene destino y peso (tiempo en minutos).

La cola de prioridad usada en el algoritmo es una version propia basada en Lista, sin usar estructuras de Java nativas. Cada elemento insertado se ordena por peso ascendente, lo que garantiza la correcta seleccion del siguiente vertice.

1. Algoritmo aplicado: Dijkstra

Dijkstra permite encontrar el camino mas corto desde un vertice origen hasta todos los demas. Funciona

manteniendo una distancia minima conocida a cada nodo y actualizandola si se encuentra un mejor camino.

El algoritmo se implementa en la clase GrafoPonderado. Se parte del nodo origen, se marcan las distancias

iniciales y se encola en la ColaConPrioridadLista. Luego se repite:

- Se desencola el vertice con menor distancia conocida

- Se recorren sus aristas

- Si el camino por ese nodo mejora el tiempo a un vecino, se actualiza la distancia y se vuelve a encolar

Esta logica se implementa sin usar HashMap ni estructuras avanzadas, solo listas y comparaciones simples.

1. Explicación del codigo clave

Fragmento clave del algoritmo (simplificado):

while (!cola.esVacia()) {

Vertice actual = cola.desencolar();

Lista<Arista> adyacentes = actual.getAdyacentes();

for (Nodo<Arista> nodoArista = adyacentes.getPrimero(); nodoArista != null; nodoArista =

nodoArista.getSiguiente()) {

Arista arista = nodoArista.getDato();

Vertice destino = arista.getDestino();

double nuevaDist = actual.getDistancia() + arista.getPeso();

if (nuevaDist < destino.getDistancia()) {

destino.setDistancia(nuevaDist);

cola.encolar(destino);

}

}

}

Este ciclo representa el corazon del algoritmo. Se extrae el vertice con menor distancia, se recorren sus vecinos, y se actualiza la distancia si se encuentra un mejor camino.

1. Diferencias con lo visto en clase

A diferencia del ejemplo en clase, este proyecto evita por completo el uso de ArrayList o PriorityQueue. Se trabaja solo con clases desarrolladas a mano: Lista, Nodo y ColaConPrioridad.

Tambien se mantiene el principio de encapsulamiento: cada clase tiene una unica responsabilidad, como manejar vertices, aristas o prioridades.

El ordenamiento de la cola se hace al insertar, comparando el valor de distancia de los vertices, garantizando eficiencia sin depender de librerias externas.

1. Comentarios adicionales

Ademas de Dijkstra, en clase se mencionaron los algoritmos de Kruskal (para arboles de expansion minima) y Floyd (para caminos entre todos los pares). Ambos podrian ser agregados en versiones del proyecto.