Actividad 3 Diseño de una red de distribución eléctrica Una empresa de energía necesita conectar varias estaciones eléctricas en una región para asegurar que toda la zona esté alimentada de manera eficiente. Las estaciones están ubicadas en diferentes ciudades y los costos de instalación de las líneas eléctricas entre ellas varían según la distancia y el terreno. Tareas: Representar el grafo utilizando una lista de adyacencia. Aplicar el algoritmo de Prim para determinar el Árbol de Recubrimiento Mínimo. Mostrar el conjunto de conexiones resultante y calcular el costo total.

Estación A -> Estación B (4)

Estación A -> Estación C (2)

Estación B -> Estación C (1)

Estación B -> Estación D (5)

Estación C -> Estación D (8)

Estación C -> Estación E (10)

Estación D -> Estación E (2)

Estación E -> Estación F (3)

Estación F -> Estación A (7)

import java.util.\*;

public class Grafico {

private Map<String, List<Conexion>> grafo = new HashMap<>();

static class Conexion {

String destino;

int costo;

Conexion(String destino, int costo) {

this.destino = destino;

this.costo = costo;

}

}

public void agregarConexion(String origen, String destino, int costo) {

grafo.computeIfAbsent(origen, k -> new ArrayList<>()).add(new Conexion(destino, costo));

grafo.computeIfAbsent(destino, k -> new ArrayList<>()).add(new Conexion(origen, costo)); // Asumiendo grafo no dirigido

}

public Map<String, List<Conexion>> obtenerGrafo() {

return grafo;

}

public static void main(String[] args) {

Grafico grafico = new Grafico();

grafico.agregarConexion("A", "B", 4);

grafico.agregarConexion("A", "C", 2);

grafico.agregarConexion("B", "C", 1);

grafico.agregarConexion("B", "D", 5);

grafico.agregarConexion("C", "D", 8);

grafico.agregarConexion("C", "E", 10);

grafico.agregarConexion("D", "E", 2);

grafico.agregarConexion("E", "F", 3);

grafico.agregarConexion("F", "A", 7);

System.out.println(grafico.obtenerGrafo());

}

}

import java.util.PriorityQueue;

public class AlgoritmoPrim {

private Set<String> visitado = new HashSet<>();

private PriorityQueue<Conexion> aristas = new PriorityQueue<>(Comparator.comparingInt(c -> c.costo));

private List<Conexion> mst = new ArrayList<>();

public List<Conexion> prim(Map<String, List<Grafico.Conexion>> grafo, String inicio) {

visitado.add(inicio);

aristas.addAll(grafo.get(inicio));

while (!aristas.isEmpty()) {

Conexion actual = aristas.poll();

if (visitado.contains(actual.destino)) continue;

visitado.add(actual.destino);

mst.add(actual);

for (Grafico.Conexion conexion : grafo.get(actual.destino)) {

if (!visitado.contains(conexion.destino)) {

aristas.add(new Conexion(conexion.destino, conexion.costo));

}

}

}

return mst;

}

public static void main(String[] args) {

Grafico grafico = new Grafico();

grafico.agregarConexion("A", "B", 4);

grafico.agregarConexion("A", "C", 2);

grafico.agregarConexion("B", "C", 1);

grafico.agregarConexion("B", "D", 5);

grafico.agregarConexion("C", "D", 8);

grafico.agregarConexion("C", "E", 10);

grafico.agregarConexion("D", "E", 2);

grafico.agregarConexion("E", "F", 3);

grafico.agregarConexion("F", "A", 7);

AlgoritmoPrim prim = new AlgoritmoPrim();

List<Conexion> resultado = prim.prim(grafico.obtenerGrafo(), "A");

int costoTotal = resultado.stream().mapToInt(c -> c.costo).sum();

System.out.println("Conjunto de conexiones resultante: " + resultado);

System.out.println("Costo total: " + costoTotal);

}

}