Actividad 2 Una empresa de viajes quiere implementar un sistema de selección de rutas que permita a los usuarios encontrar el viaje más barato para llegar a un destino, incluso realizando escalas. Cada ciudad se representa como un nodo en un grafo dirigido, y cada ruta aérea entre dos ciudades tiene un costo asociado. El sistema debe encontrar el camino más barato desde una ciudad de origen hasta una ciudad destino, permitiendo escalas si es necesario. Permitir al usuario ingresar una ciudad de origen y una de destino. Buscar la ruta más barata utilizando el algoritmo Uniform Cost Search (UCS). Mostrar el costo mínimo y el itinerario de escalas.

**Pasos para Implementar el Sistema**

1. **Representación del Grafo**:
   * Cada ciudad es un nodo.
   * Cada ruta aérea es una arista dirigida con un costo asociado.
   * El grafo se puede representar como un diccionario de listas de adyacencia en Python.
2. **Algoritmo Uniform Cost Search (UCS)**:
   * UCS es una variante del algoritmo de búsqueda en amplitud (BFS) que tiene en cuenta los costos de las aristas.
   * Utiliza una cola de prioridad (heap) para explorar siempre el nodo con el menor costo acumulado.
   * Mantiene un registro de los costos mínimos para cada nodo y evita ciclos.
3. **Entrada del Usuario**:
   * El usuario ingresa la ciudad de origen y la ciudad de destino.
4. **Salida del Sistema**:
   * Muestra el costo mínimo y el itinerario de escalas

import java.util.\*;

// Interfaz para representar un grafo

interface Grafo {

Map<String, List<Arista>> obtenerAdyacencias();

}

// Clase que implementa la interfaz Grafo

class GrafoConcreto implements Grafo {

private Map<String, List<Arista>> adyacencias;

public GrafoConcreto() {

adyacencias = new HashMap<>();

}

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

adyacencias.putIfAbsent(origen, new ArrayList<>());

adyacencias.get(origen).add(new Arista(destino, costo));

}

@Override

public Map<String, List<Arista>> obtenerAdyacencias() {

return adyacencias;

}

}

// Clase para representar una arista (ruta entre dos ciudades)

class Arista {

String destino;

int costo;

public Arista(String destino, int costo) {

this.destino = destino;

this.costo = costo;

}

}

// Clase para realizar la búsqueda de costo uniforme

class BusquedaCostoUniforme {

private Grafo grafo;

public BusquedaCostoUniforme(Grafo grafo) {

this.grafo = grafo;

}

public ResultadoBusqueda buscarRutaMasBarata(String origen, String destino) {

// Cola de prioridad para UCS

PriorityQueue<Nodo> cola = new PriorityQueue<>(Comparator.comparingInt(n -> n.costoAcumulado));

cola.add(new Nodo(origen, 0, Arrays.asList(origen)));

Set<String> visitados = new HashSet<>();

while (!cola.isEmpty()) {

Nodo actual = cola.poll();

if (actual.ciudad.equals(destino)) {

return new ResultadoBusqueda(actual.costoAcumulado, actual.ruta);

}

if (!visitados.contains(actual.ciudad)) {

visitados.add(actual.ciudad);

for (Arista arista : grafo.obtenerAdyacencias().getOrDefault(actual.ciudad, new ArrayList<>())) {

if (!visitados.contains(arista.destino)) {

List<String> nuevaRuta = new ArrayList<>(actual.ruta);

nuevaRuta.add(arista.destino);

cola.add(new Nodo(arista.destino, actual.costoAcumulado + arista.costo, nuevaRuta));

}

}

}

}

return null; // No se encontró una ruta

}

}

// Clase para representar un nodo en la búsqueda

class Nodo {

String ciudad;

int costoAcumulado;

List<String> ruta;

public Nodo(String ciudad, int costoAcumulado, List<String> ruta) {

this.ciudad = ciudad;

this.costoAcumulado = costoAcumulado;

this.ruta = ruta;

}

}

// Clase para representar el resultado de la búsqueda

class ResultadoBusqueda {

int costoMinimo;

List<String> ruta;

public ResultadoBusqueda(int costoMinimo, List<String> ruta) {

this.costoMinimo = costoMinimo;

this.ruta = ruta;

}

@Override

public String toString() {

return "Costo mínimo: " + costoMinimo + "\nItinerario: " + String.join(" -> ", ruta);

}

}

// Clase principal para ejecutar el programa

public class SistemaViajes {

public static void main(String[] args) {

// Crear el grafo

GrafoConcreto grafo = new GrafoConcreto();

grafo.agregarRuta("A", "B", 10);

grafo.agregarRuta("A", "C", 15);

grafo.agregarRuta("B", "D", 12);

grafo.agregarRuta("B", "E", 15);

grafo.agregarRuta("C", "F", 10);

grafo.agregarRuta("D", "G", 5);

grafo.agregarRuta("E", "G", 20);

grafo.agregarRuta("F", "G", 10);

// Crear el buscador

BusquedaCostoUniforme buscador = new BusquedaCostoUniforme(grafo);

// Entrada del usuario

Scanner scanner = new Scanner(System.in);

System.out.print("Ingrese la ciudad de origen: ");

String origen = scanner.nextLine().trim().toUpperCase();

System.out.print("Ingrese la ciudad de destino: ");

String destino = scanner.nextLine().trim().toUpperCase();

// Buscar la ruta más barata

ResultadoBusqueda resultado = buscador.buscarRutaMasBarata(origen, destino);

// Mostrar el resultado

if (resultado != null) {

System.out.println("\n" + resultado);

} else {

System.out.println("\nNo se encontró una ruta válida.");

}

}

}