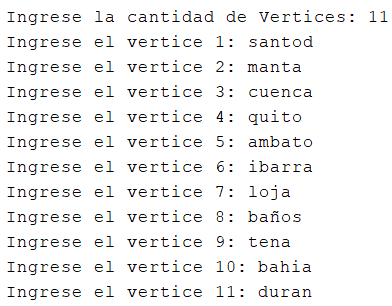
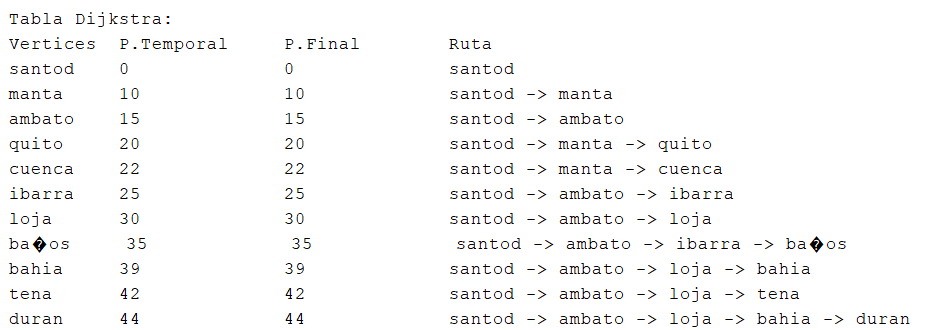
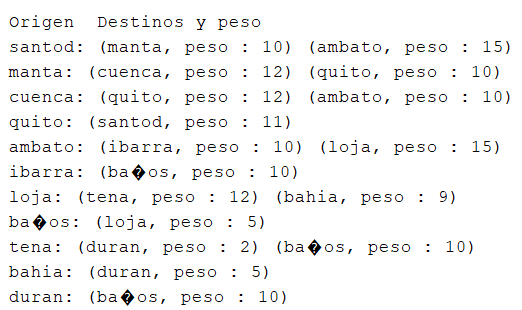
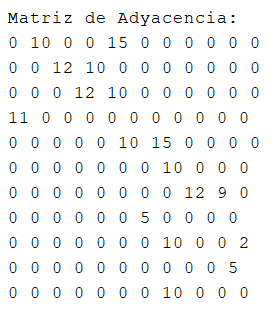
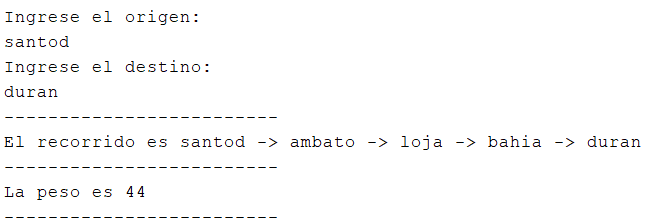
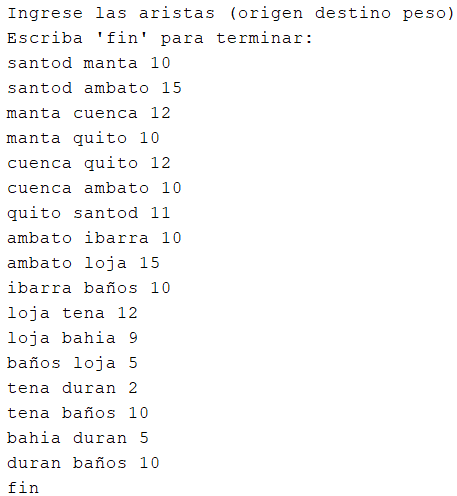
|  |
| --- |
| **Main**  public static void main(String[] args) {  //Instanciamos la clase Scanneer para obteener entrada del usuario  Scanner scanner = new Scanner(System.in);  // Solicitar al usuario la cantidad de vértices/nodos en el grafo  System.out.print("Ingrese la cantidad de Vertices: ");  int cantidadVertices = scanner.nextInt();  // Crear un objeto Grafo con la cantidad de vértices proporcionada  List<String> vertices = new ArrayList<>();  for (int i = 0; i < cantidadVertices; i++) {  System.out.print("Ingrese el vertice " + (i + 1) + ": ");  vertices.add(scanner.next());  }  Grafo grafo = new Grafo(vertices);  // Solicitar al usuario ingresar las aristas del grafo (origen destino peso)  System.out.println("Ingrese las aristas (origen destino peso), escriba 'fin' para terminar:");  System.out.println("Escriba 'fin' para terminar:");  while (true) {  // Leer la entrada del usuario  String origen = scanner.next();  // Salir del bucle si el usuario escribe 'fin'  if (origen.equalsIgnoreCase("fin")) {  break;  }  // Convertir las entradas a valores numéricos y agregar la arista al grafo  String destino = scanner.next();  int peso = scanner.nextInt();  grafo.agregarArista(origen, destino, peso);  }  // Solicitar al usuario ingresar el vértice de origen para el algoritmo de Dijkstra  System.out.println("Ingrese el origen: ");  String origenDijkstra = scanner.next();  // Solicitar al usuario ingresar el vértice de destino para el algoritmo de Dijkstra  System.out.println("Ingrese el destino: ");  String destinoDijkstra = scanner.next();  // Aplicar el algoritmo de Dijkstra y obtener el resultado  List<String> dijkstraResult = grafo.dijkstra(origenDijkstra, destinoDijkstra);  // Imprimir el recorrido resultante del algoritmo de Dijkstra  System.out.println("-------------------------");  System.out.println("El recorrido es " + dijkstraResult.stream().collect(Collectors.joining(" -> ")));  System.out.println("-------------------------");  // Imprimir la suma total de los pesos de las aristas en el recorrido  System.out.println("La peso es " + grafo.sumaPesosAristas(dijkstraResult));  System.out.println("-------------------------");  // Mostrar la matriz de adyacencia del grafo  grafo.mostrarMatrizAdyacencia();  System.out.println("-------------------------");  //Mostrar nodos con sus vecinos y su peso  grafo.mostrarVecinos();  System.out.println("-------------------------");  grafo.mostrarTablaDijkstra(origenDijkstra, destinoDijkstra);}  } |
| **Grafo**  public class Grafo {  private final Map<String, Integer> indiceVertices;  private final List<List<Arista>> listaAdyacencia;  // Constructor que inicializa la lista de adyacencia y el mapeo de índices para los vértices.  public Grafo(List<String> vertices) {  int cantidadVertices = vertices.size();  this.listaAdyacencia = new ArrayList<>(cantidadVertices);  this.indiceVertices = new HashMap<>();  // Inicializa la lista de adyacencia y asigna un índice a cada vértice.  for (int i = 0; i < cantidadVertices; i++) {  this.listaAdyacencia.add(new ArrayList<>());  this.indiceVertices.put(vertices.get(i), i);  }  }  // Agrega una arista entre dos vértices con un peso dado a la lista de adyacencia.  public void agregarArista(String origen, String destino, int peso) {  int indiceOrigen = indiceVertices.get(origen);  int indiceDestino = indiceVertices.get(destino);  this.listaAdyacencia.get(indiceOrigen).add(new Arista(indiceDestino, peso));  }  // Implementa el algoritmo de Dijkstra para encontrar la ruta más corta desde un nodo de origen hasta un nodo de destino.  public List<String> dijkstra(String origen, String destino) {  int indiceOrigen = indiceVertices.get(origen);  int indiceDestino = indiceVertices.get(destino);  PriorityQueue<Nodo> colaPrioridad = new PriorityQueue<>(listaAdyacencia.size(), Comparator.comparingInt(a -> a.peso));  int[] distancias = new int[listaAdyacencia.size()];  int[] previos = new int[listaAdyacencia.size()];  Arrays.fill(distancias, Integer.MAX\_VALUE);  Arrays.fill(previos, -1);  colaPrioridad.add(new Nodo(indiceOrigen, 0));  distancias[indiceOrigen] = 0;  while (!colaPrioridad.isEmpty()) {  Nodo nodoActual = colaPrioridad.poll();  int u = nodoActual.vertice;  for (Arista arista : listaAdyacencia.get(u)) {  int v = arista.destino;  int peso = arista.peso;  if (distancias[u] + peso < distancias[v]) {  distancias[v] = distancias[u] + peso;  previos[v] = u;  colaPrioridad.add(new Nodo(v, distancias[v]));  }  }  }  List<String> ruta = new ArrayList<>();  for (int i = indiceDestino; i != -1; i = previos[i]) {  ruta.add(obtenerVerticePorIndice(i));  }  Collections.reverse(ruta);  return ruta;  }  // Calcula la suma de los pesos de las aristas en la ruta proporcionada.  public int sumaPesosAristas(List<String> ruta) {  int suma = 0;  for (int i = 0; i < ruta.size() - 1; i++) {  String origen = ruta.get(i);  String destino = ruta.get(i + 1);  int peso = obtenerPesoArista(origen, destino);  suma += peso;  }  return suma;  }  // Obtiene el peso de la arista entre dos vértices dados.  private int obtenerPesoArista(String origen, String destino) {  int indiceOrigen = indiceVertices.get(origen);  int indiceDestino = indiceVertices.get(destino);  for (Arista arista : listaAdyacencia.get(indiceOrigen)) {  if (arista.destino == indiceDestino) {  return arista.peso;  }  }  return Integer.MAX\_VALUE;  }  // Muestra la matriz de adyacencia para fines de depuración.  public void mostrarMatrizAdyacencia() {  System.out.println("Matriz de Adyacencia:");  for (int i = 0; i < listaAdyacencia.size(); i++) {  for (int j = 0; j < listaAdyacencia.size(); j++) {  int peso = obtenerPesoArista(obtenerVerticePorIndice(i), obtenerVerticePorIndice(j));  System.out.print((peso == Integer.MAX\_VALUE ? "0" : peso) + " ");  }  System.out.println();  }  }  // Muestra los vecinos de cada nodo junto con los pesos de las aristas.  public void mostrarVecinos() {  System.out.println("Origen Destinos y peso ");  for (int i = 0; i < listaAdyacencia.size(); i++) {  String nodo = obtenerVerticePorIndice(i);  System.out.print(nodo + ": ");  for (Arista arista : listaAdyacencia.get(i)) {  String vecino = obtenerVerticePorIndice(arista.destino);  int peso = arista.peso;  System.out.print("(" + vecino + ", peso : " + peso + ") ");  }  System.out.println();  }  }  // Muestra la tabla dijkstra  public void mostrarTablaDijkstra(String origen, String destino) {  int indiceOrigen = indiceVertices.get(origen);  int indiceDestino = indiceVertices.get(destino);  System.out.println("Tabla Dijkstra:");  System.out.printf("%-10s%-15s%-15s%-15s%n", "Vertices", "P.Temporal", "P.Final", "Ruta");  PriorityQueue<Nodo> colaPrioridad = new PriorityQueue<>(listaAdyacencia.size(), Comparator.comparingInt(a -> a.peso));  int[] distancias = new int[listaAdyacencia.size()];  int[] previos = new int[listaAdyacencia.size()];  boolean[] finalizados = new boolean[listaAdyacencia.size()];  Arrays.fill(distancias, Integer.MAX\_VALUE);  Arrays.fill(previos, -1);  colaPrioridad.add(new Nodo(indiceOrigen, 0));  distancias[indiceOrigen] = 0;  while (!colaPrioridad.isEmpty()) {  Nodo nodoActual = colaPrioridad.poll();  int u = nodoActual.vertice;  if (!finalizados[u]) {  finalizados[u] = true;  for (Arista arista : listaAdyacencia.get(u)) {  int v = arista.destino;  int peso = arista.peso;  if (distancias[u] + peso < distancias[v]) {  distancias[v] = distancias[u] + peso;  previos[v] = u;  colaPrioridad.add(new Nodo(v, distancias[v]));  }  }  String camino = obtenerCamino(previos, indiceOrigen, u);  System.out.printf("%-10s%-15s%-15s%-15s%n",  obtenerVerticePorIndice(u),  distancias[u] == Integer.MAX\_VALUE ? "Infinito" : distancias[u],  finalizados[u] ? (distancias[u] == Integer.MAX\_VALUE ? "Infinito" : distancias[u]) : "No",  camino);  }  }  }  // Obtiene el camino desde el origen hasta el vértice actual.  private String obtenerCamino(int[] previos, int origen, int destino) {  List<String> camino = new ArrayList<>();  for (int i = destino; i != -1; i = previos[i]) {  camino.add(obtenerVerticePorIndice(i));  }  Collections.reverse(camino);  return String.join(" -> ", camino);  }  // Obtiene el vértice correspondiente a un índice dado en el mapeo de índices.  private String obtenerVerticePorIndice(int indice) {  for (Map.Entry<String, Integer> entry : indiceVertices.entrySet()) {  if (entry.getValue() == indice) {  return entry.getKey();  }  }  return null;  }  // Clase interna que representa un nodo  private static class Nodo {  private final int vertice;  private final int peso;  public Nodo(int vertice, int peso) {  this.vertice = vertice;  this.peso = peso;  }  }  // Clase interna que representa una arista  private static class Arista {  private final int destino;  private final int peso;  public Arista(int destino, int peso) {  this.destino = destino;  this.peso = peso;}}  } |

El código pide el ingreso de la cantidad de vértices y después se ingresa cada vértice que son las ciudades. Luego de ingresar se ingresa la ciudad de origen con la ciudad de destino y su peso que son los kilómetros de distancia. Una vez ingresado nos pide la ciudad de origen y la ciudad de destino hacia donde va. Calcula la ruta que se realiza mostrándola y con los kilómetros(peso) a recorrer. También muestra la lista y la matriz dijtra con los datos realizados para obtener el peso y la ruta o recorrido.



Pide el ingreso de los vértices que se ingresaran como origen y destino que vienen siendo las ciudades de ecuador



Muestra la matriz de adyacencia con el peso o kilómetros de distancia

Se ingresa el origen y destino según el vértice y el peso que son los kilómetros de distancia

El peso vendría siendo los kilómetros a recorrer durante el viaje

Te pide la ciudad de origen, de donde te encuentras

Te pide la ciudad de destino, hacia dónde vas

Y te muestra la ruta que vas a realizar para llegar a tu destino y el peso

Muestra la ciudad de origen y los destino que tiene con sus Km o peso

Muestra la tabla donde están las ciudades y el recorrido que hace para obtener la ruta desde santod hasta la ciudad de duran donde nos muestra los kilómetros (peso) que se recorre





