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.

```
Ingrese la cantidad de Vertices: 11
Ingrese el vertice 1: santod
Ingrese el vertice 2: manta
Ingrese el vertice 3: cuenca
Ingrese el vertice 4: quito
Ingrese el vertice 5: ambato
Ingrese el vertice 6: ibarra
Ingrese el vertice 7: loja
Ingrese el vertice 8: baños
Ingrese el vertice 9: tena
Ingrese el vertice 10: bahia
Ingrese el vertice 11: duran
```

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

```
Ingrese las aristas (origen destino peso)
Escriba 'fin' para terminar:
santod manta 10
santod ambato 15
manta cuenca 12
manta quito 10
cuenca quito 12
cuenca ambato 10
quito santod 11
ambato ibarra 10
ambato loja 15
ibarra baños 10
loja tena 12
loja bahia 9
baños loja 5
tena duran 2
tena baños 10
bahia duran 5
duran baños 10
fin
Ingrese el origen:
santod
Ingrese el destino:
duran
----------------------
El recorrido es santod -> ambato -> loja -> bahia -> duran
--------------------------
La peso es 44
--------------------------
Matriz de Adyacencia:
0 10 0 0 15 0 0 0 0 0 0
0 0 12 10 0 0 0 0 0 0 0
0 0 0 12 10 0 0 0 0 0 0
11 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 10 15 0 0 0 0
0 0 0 0 0 0 0 10 0 0 0
0 0 0 0 0 0 0 0 12 9 0
0 0 0 0 0 0 5 0 0 0 0
0 0 0 0 0 0 0 10 0 0 2
0 0 0 0 0 0 0 0 0 0 5
0 0 0 0 0 0 0 10 0 0 0
```

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

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

El peso vendría siendo los kilómetros a recorrer durante el viaje

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

```
Origen  Destinos y peso
santod: (manta, peso : 10) (ambato, peso : 15)
manta: (cuenca, peso : 12) (quito, peso : 10)
cuenca: (quito, peso : 12) (ambato, peso : 10)
quito: (santod, peso : 11)
ambato: (ibarra, peso : 10) (loja, peso : 15)
ibarra: (ba�os, peso : 10)
loja: (tena, peso : 12) (bahia, peso : 9)
ba�os: (loja, peso : 5)
tena: (duran, peso : 2) (ba�os, peso : 10)
bahia: (duran, peso : 5)
duran: (ba�os, peso : 10)
```

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

```
Tabla Dijkstra:
Vertices  P.Temporal     P.Final        Ruta
santod    0              0              santod
manta     10             10             santod -> manta
ambato    15             15             santod -> ambato
quito     20             20             santod -> manta -> quito
cuenca    22             22             santod -> manta -> cuenca
ibarra    25             25             santod -> ambato -> ibarra
loja      30             30             santod -> ambato -> loja
ba�os      35             35             santod -> ambato -> ibarra -> ba�os
bahia     39             39             santod -> ambato -> loja -> bahia
tena      42             42             santod -> ambato -> loja -> tena
duran     44             44             santod -> ambato -> loja -> bahia -> duran
```

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

```java
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<>( initialCapacity: 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( key: vertices.get( index: i),  value: 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( key: origen);
        int indiceDestino = indiceVertices.get( key: destino);
        this.listaAdyacencia.get( index: indiceOrigen).add(new Arista( destino: indiceDestino, peso));
    }

    // Implementa el algoritmo de Dijkstra para encontrar la ruta más corta desde un nodo de origen
    public List<String> dijkstra(String origen, String destino) {
        int indiceOrigen = indiceVertices.get( key: origen);
        int indiceDestino = indiceVertices.get( key: destino);

        PriorityQueue<Nodo> colaPrioridad = new PriorityQueue<>( initialCapacity: listaAdyacencia.size(),
        int[] distancias = new int[listaAdyacencia.size()];
        int[] previos = new int[listaAdyacencia.size()];
```

```java
public void mostrarTablaDijkstra(String origen, String destino) {
    int indiceOrigen = indiceVertices.get( key: origen);
    int indiceDestino = indiceVertices.get( key: destino);

    System.out.println( x: "Tabla Dijkstra:");
    System.out.printf( format: "%-10s%-15s%-15s%-15s%n",  args: "Vertices",  args: "P.Temporal",

    PriorityQueue<Nodo> colaPrioridad = new PriorityQueue<>( initialCapacity: listaAdyacencia.
    int[] distancias = new int[listaAdyacencia.size()];
    int[] previos = new int[listaAdyacencia.size()];
    boolean[] finalizados = new boolean[listaAdyacencia.size()];
    Arrays.fill( a: distancias,  val: Integer.MAX_VALUE);
    Arrays.fill( a: previos,  val: -1);

    colaPrioridad.add(new Nodo( vertice: indiceOrigen,  peso: 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( index: 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( vertice: v, distancias[v]));
                }
            }

            String camino = obtenerCamino(previos,  origen: indiceOrigen,  destino: u);
```

```java
// 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( e: obtenerVerticePorIndice( indice: i));
    }
    Collections.reverse( list: camino);
    return String.join( delimiter: " -> ",  elements: camino);
}

// Obtiene el vértice correspondiente a un índice dado en el mapeo de ín
private String obtenerVerticePorIndice(int indice) {
    for (Map.Entry<String, Integer> entry : indiceVertices.entrySet()) {
        if (entry.getValue() == indice) {
            return entry.getKey();
        }
    }
    return null;
}
public static void main(String[] args) {
      //Instanciamos la clase Scanneer para obteener entrada del us
    Scanner scanner = new Scanner( source: System.in);
    // Solicitar al usuario la cantidad de vértices/nodos en el gra
    System.out.print( s: "Ingrese la cantidad de Vertices: ");
    int cantidadVertices = scanner.nextInt();
    // Crear un objeto Grafo con la cantidad de vértices proporcio
    List<String> vertices = new ArrayList<>();
    for (int i = 0; i < cantidadVertices; i++) {
        System.out.print("Ingrese el vertice " + (i + 1) + ": ");
        vertices.add( e: scanner.next());
    }
    Grafo grafo = new Grafo(vertices);
    // Solicitar al usuario ingresar las aristas del grafo (origen
    System.out.println( x: "Ingrese las aristas (origen destino peso)
    System.out.println( x: "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( anotherString: "fin")) {
            break;
        }
        // Convertir las entradas a valores numéricos y agregar la
        String destino = scanner.next();
        int peso = scanner.nextInt();
        grafo.agregarArista(origen, destino, peso);
```

```java
// Solicitar al usuario ingresar el vértice de destino para el algoritmo de Di
System.out.println( x: "Ingrese el destino: ");
String destinoDijkstra = scanner.next();

// Aplicar el algoritmo de Dijkstra y obtener el resultado
List<String> dijkstraResult = grafo.dijkstra( origen: origenDijkstra,  destino: desti
// Imprimir el recorrido resultante del algoritmo de Dijkstra
System.out.println("El recorrido es " + dijkstraResult.stream().collect( collectc
System.out.println( x: "-------------------------");
// Imrimir la suma total de los pesos de las aristas en el recorrido
System.out.println("La peso es " + grafo.sumaPesosAristas( ruta: dijkstraResult))
System.out.println( x: "-------------------------");
// Mostrar la matriz de adyacencia del grafo
grafo.mostrarMatrizAdyacencia();
//Mostrar nodos con sus vecinos y su peso
grafo.mostrarVecinos();
grafo.mostrarTablaDijkstra( origen: origenDijkstra,  destino: destinoDijkstra);}
```