Codigo

Clase Grafo

```
import java.util.ArrayList;
import java.util.Arrays;
import java.util.Collections;
import java.util.Comparator;
import java.util.List;
import java.util.PriorityQueue;
/**
 *
 * @author gisse
 */
public class Grafo {
   //matriz de adyacencia
   private final int cantidadVertices;//creo una lista
   private final List<List<Arista>> listaAdyacencia;
   //metodo para inicializar variables
   public Grafo(int cantidadVertices) {
      this.cantidadVertices = cantidadVertices;
      this.listaAdyacencia = new ArrayList<>(cantidadVertices);
      for (int i = 0; i < cantidadVertices; i++) {
         this.listaAdyacencia.add(new ArrayList<>());
      }
   }
//constructor para definir e inicializar las variables
   public void agregarArista(char origen, char destino, int peso) {
      this.listaAdyacencia.get(origen - 'A').add(new Arista(destino - 'A', peso));//definir letras
   }
   //metodo para crear el dijkstra
   public List<Character> dijkstra(char origen, char destino) {
      PriorityQueue<Nodo> colaPrioridad = new PriorityQueue<>(cantidadVertices,
Comparator.comparingInt(a -> a.peso));
      int[] distancias = new int[cantidadVertices];
      int[] previos = new int[cantidadVertices];//arreglo para cantidad de vertices
      Arrays.fill(distancias, Integer.MAX_VALUE);
      Arrays.fill(previos, -1);
      colaPrioridad.add(new Nodo(origen - 'A', 0));
      distancias[origen - 'A'] = 0;//define origen
      while (!colaPrioridad.isEmpty()) {
         Nodo nodoActual = colaPrioridad.poll();
         int u = nodoActual.vertice;
//para ver cantidad de aritas e ingresar dartos
         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]));
            }
         }
```

```
      }
//lista de modo carcter busca la ruta y el destino
      List<Character> ruta = new ArrayList<>();
      for (int i = destino - 'A'; i != -1; i = previos[i]) {
         ruta.add((char) (i + 'A'));
      }
      Collections.reverse(ruta);
      return ruta; }
//metodo para vertice y peso
   private static class Nodo {
      private final int vertice;
      private final int peso;
//nodo para el vertice y el peso
      public Nodo(int vertice, int peso) {
         this.vertice = vertice;
         this.peso = peso;
      }}
//metodo para realizar la suma y saber el peso del grafo y la ruta
   public int sumaPesosAristas(List<Character> ruta) {
      int suma = 0;
      for (int i = 0; i < ruta.size() - 1; i++) {
         char origen = ruta.get(i);
         char destino = ruta.get(i + 1);
         int peso = obtenerPesoArista(origen, destino);
         suma += peso; }
      return suma; }
//metodo para mostrar la matriz de adyacencia
   public void mostrarMatrizAdyacencia() {
      System.out.println("Matriz de Adyacencia es ");
      for (int i = 0; i < cantidadVertices; i++) {
         for (int j = 0; j < cantidadVertices; j++) {
            int peso = obtenerPesoArista((char) ('A' + i), (char) ('A' + j));
            System.out.print((peso == Integer.MAX_VALUE ? "0" : peso) + " ");
         }
         System.out.println();
      } }
//metodo para obtener el peso de las aristas
   private int obtenerPesoArista(char origen, char destino) {
      for (Arista arista : listaAdyacencia.get(origen - 'A')) {
         if (arista.destino == destino - 'A') {
            return arista.peso;
         }
      }
      return Integer.MAX_VALUE;
   }
//metodo para inicializar las aristas
   private static class Arista {
      private final int destino;
      private final int peso;
//constructor del metodo arustas
      public Arista(int destino, int peso) {
         this.destino = destino;
```

```
        this.peso = peso;
    }
}}
```

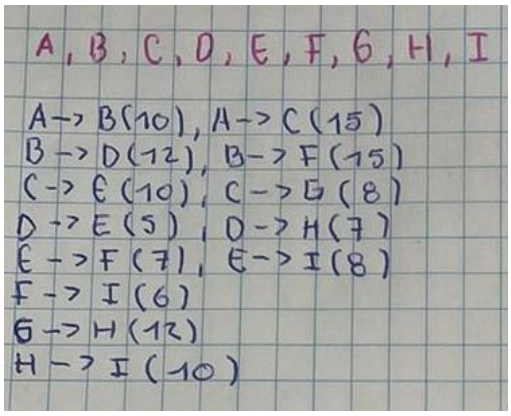
Clase main

```
public class GrafoBusqueda {
    public static void main(String[] args) {
        // TODO code application logic here
       Grafo grafo = new Grafo(9); // Por ejemplo, un grafo con 6 vértices

        // Agregar aristas al grafo con origen destino y peso
        grafo.agregarArista('A', 'B', 10);
        grafo.agregarArista('A', 'C', 15);
        grafo.agregarArista('B', 'D', 12);
        grafo.agregarArista('B', 'F', 15);
        grafo.agregarArista('C', 'E', 10);
        grafo.agregarArista('C', 'G', 8);
        grafo.agregarArista('D', 'E', 5);
        grafo.agregarArista('D', 'H', 7);
        grafo.agregarArista('E', 'F', 7);
        grafo.agregarArista('E', 'I', 8);
        grafo.agregarArista('F', 'I', 6);
        grafo.agregarArista('G', 'H', 12);
        grafo.agregarArista('H', 'I', 10);
//muestra el recorrido del grafo
        System.out.println("EL RECORRIDO ES " + grafo.dijkstra('A', 'I'));
//mustra el peso del grafo
        System.out.println("EL PESO ES " + grafo.sumaPesosAristas(grafo.dijkstra('A', 'I')));
//muestra la matriz de adyacencia del garfo
        grafo.mostrarMatrizAdyacencia();
    }
}
```

```java
        // Agregar aristas al grafo
        grafo.agregarArista( origen: 'A',  destino: 'B',  peso: 10);
        grafo.agregarArista( origen: 'A',  destino: 'C',  peso: 15);
        grafo.agregarArista( origen: 'B',  destino: 'D',  peso: 12);
        grafo.agregarArista( origen: 'B',  destino: 'F',  peso: 15);
        grafo.agregarArista( origen: 'C',  destino: 'E',  peso: 10);
        grafo.agregarArista( origen: 'C',  destino: 'G',  peso: 8);
        grafo.agregarArista( origen: 'D',  destino: 'E',  peso: 5);
        grafo.agregarArista( origen: 'D',  destino: 'H',  peso: 7);
        grafo.agregarArista( origen: 'E',  destino: 'F',  peso: 7);
        grafo.agregarArista( origen: 'E',  destino: 'I',  peso: 8);
        grafo.agregarArista( origen: 'F',  destino: 'I',  peso: 6);
        grafo.agregarArista( origen: 'G',  destino: 'H',  peso: 12);
        grafo.agregarArista( origen: 'H',  destino: 'I',  peso: 10);

        System.out.println("EL RECORRIDO ES " + grafo.dijkstra( origen: 'A',  destino: 'I'));
        System.out.println("EL PESO ES " + grafo.sumaPesosAristas( ruta: grafo.dijkstra( origen: 'A',  destino: 'I')));
        grafo.mostrarMatrizAdyacencia();
    }
```

```
- Grafo_Lab1U3 (run) ×
run:
EL RECORRIDO ES [A, B, F, I]
EL PESO ES 31
Matriz de Adyacencia es
0 10 15 0 0 0 0 0 0
0 0 0 12 0 15 0 0 0
0 0 0 0 10 0 8 0 0
0 0 0 0 5 0 0 7 0
0 0 0 0 0 7 0 0 8
0 0 0 0 0 0 0 0 6
0 0 0 0 0 0 0 12 0
0 0 0 0 0 0 0 0 10
0 0 0 0 0 0 0 0 0
```

```
utput - Grafo_Lab1U3 (run) ×
run:
LISTA
Lista de adyacencia:
A -> 1 Peso: 10
A -> 2 Peso: 15
B -> 3 Peso: 12
B -> 5 Peso: 15
C -> 4 Peso: 10
C -> 6 Peso: 8
D -> 4 Peso: 5
D -> 7 Peso: 7
E -> 5 Peso: 7
E -> 8 Peso: 8
F -> 8 Peso: 6
G -> 7 Peso: 12
H -> 8 Peso: 10
```



t - Grafo_Lab1U3 (run) ×

```
run:
EL RECORRIDO ES [A, B, F, I]
EL PESO ES 31
Matriz de Adyacencia es
0 10 15 0 0 0 0 0 0
0 0 0 12 0 15 0 0 0
0 0 0 0 10 0 8 0 0
0 0 0 0 5 0 0 7 0
0 0 0 0 0 7 0 0 8
0 0 0 0 0 0 0 0 6
0 0 0 0 0 0 0 12 0
0 0 0 0 0 0 0 0 10
0 0 0 0 0 0 0 0 0
BUILD SUCCESSFUL (total time:
```

Nombre: Gisela Saldarriaga Salazar Fecha: 22/02/24
Ing
Curso: Materia

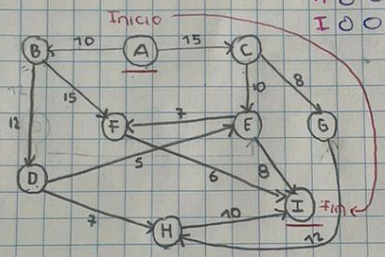
Definición
Se tiene un grafo dirigido que representa una red entre ~~dos~~ ciudades cada arista representa una ruta entre dos ciudade con un peso que representa la distancia de ellas. El objetivo es encontra la ruta mas corta de una ciudad de origen a su destino.

A, B, C, D, E, F, G, H, I

A -> B(10), A -> C(15)
B -> D(12), B -> F(15)
C -> E(10), C -> G(8)
D -> E(5), D -> H(7)
E -> F(7), E -> I(8)
F -> I(6)
G -> H(12)
H -> I(10)

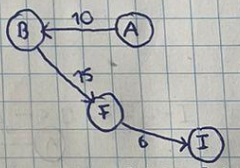
Matriz Ad.

	A	B	C	D	E	F	G	H	I
A	0	10	15	0	0	0	0	0	0
B	0	0	0	12	0	15	0	0	0
C	0	0	0	0	10	0	8	0	0
D	0	0	0	0	5	0	0	7	0
E	0	0	0	0	0	7	0	0	8
F	0	0	0	0	0	0	0	0	6
G	0	0	0	0	0	0	0	12	0
H	0	0	0	0	0	0	0	0	10
I	0	0	0	0	0	0	0	0	0



(2)

A - I



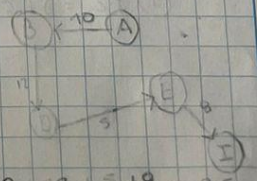
10+15+6 = 31

#	Final	Temporal	
A	0	0	
B	10	10	
C	15	15	
D	22	22	10+12 = 22
E	27	27	22+5 = 27
F	25	25	10+15 = 25 | 25+7 = 32
G	23	23	
H	29	29	22+7 = 29
I	31	31	27+8 = 35 25+6 = 31

Ruta = A -> B -> F -> I

Ruta = A -> B -> D -> H -> I

24/02/22



10 12 5 8 = 35