UNIVERSIDAD NACIONAL DE SAN AGUSTÍN DE AREQUIPA
FACULTAD DE PRODUCCIÓN Y SERVICIOS
ESCUELA PROFESIONAL DE INGENIERÍA DE SISTEMAS

LAB EDA GRUPO B

ACTIVIDAD: "GRAFOS"

DOCENTE: EDITH PAMELA RIVERO TUPAC

ALUMNO:

- Chávez Chambi, Marco David

AREQUIPA – PERÚ
2021

EJERCICIOS PROPUESTOS

1. Crear un repositorio en GitHub, donde incluyan la resolución de los ejercicios propuestos y el informe.

2. Implementar el cogido de Grafo cuya representación sea realizada mediante LISTA DE ADYACENCIA.

```java
import java.util.ArrayList;
import java.util.List;

public class Grafo {
    private List<Vertice> vertices;
    private List<Arco> listAd;

    public Grafo() {
        this.vertices = new ArrayList<>();
        this.listAd = new ArrayList<>();
    }

    public void insertar(String dato) {
        Vertice nuevoVertice = new Vertice(dato);
        vertices.add(nuevoVertice);

    }

    public void eliminar(String dato) {
        for (Vertice vertice : vertices) {
            if(vertice.getDato().contentEquals(dato)) {
                for(Arco arco : listAd) {
                    if(arco.getOrigen().getDato().equals(dato)) {
                        arco = null;
                    }
                    if(arco.getDestino().getDato().equals(dato)) {
                        arco = null;
                    }
                }
                vertices.remove(vertice);
            }
        }
    }

    public void buscar() {

    }

    public void listar() {

    }



}
```

3. Implementar BSF, DFS y Dijkstra con sus respectivos casos de prueba.

```java
public class Dijkstra {
    public static void dijkstra(int[][] grafo, int vertR){
        int cantVertice = grafo.length;
        boolean[] visitaVertice = new boolean[cantVertice];
        int[] distancia = new int[cantVertice];
        for (int i = 0; i < cantVertice; i++){
            visitaVertice[i] = false;
            distancia[i] = Integer.MAX_VALUE;
        }
        distancia[vertR] = 0;
        for (int i = 0; i < cantVertice; i++){

            int u = buscarDistMinima(distancia, visitaVertice);
            visitaVertice[u] = true;
            for (int v =0 ; v < cantVertice; v++){
                if(!visitaVertice[v] && grafo[u][v] != 0 && (distancia[u] + grafo[u][v] < distancia[v])){
                    distancia[v] = distancia[u] + grafo[u][v];
                }
            }
        }
        for (int i = 0; i < distancia.length; i++){
            System.out.println(String.format("Distancia del vértice origen al vértice fuente %s al vértice %s es %s", vertR, i, distancia[i]));
        }

    }

    private static int buscarDistMinima(int[] distancia, boolean[] visitaVertice) {
        int minDist = Integer.MAX_VALUE;
        int minDistVertice = -1;
        for (int i =0; i < distancia.length; i++){
            if(!visitaVertice[i] && distancia[i] < minDist){
                minDist = distancia[i];
                minDistVertice = i;
            }
        }
        return minDistVertice;
    }

    public static void main(String[] args) {
        int grafo1[][] = new int[][] {
                { 0, 4, 8, 0, 0 },
                { 4, 0, 2, 5, 0 },
                { 8, 2, 0, 5, 9},
                { 0, 5, 5, 0, 4 },
                { 0, 0, 9, 4, 0 }
        };
        Dijkstra t = new Dijkstra();
        t.dijkstra(grafo1, 0);
    }
}
```

CUESTIONARIO

1. ¿Cuantas variantes del algoritmo de Dijkstra hay y cuál es la diferencia entre ellas?

Se encontraron 2 variantes, estas se ejecutan rápidamente pero no calculan correctamente las rutas más cortas cuando hay bordes negativos. Otras variantes siempre calculan los caminos más cortos correctamente, pero pueden requerir un tiempo exponencial en el peor de los casos si hay bordes negativos

2. Investigue sobre los ALGORITMOS DE CAMINOS MINIMOS e indique, ¿Qué similitudes encuentra, qué diferencias, en qué casos utilizar y por qué?

- El algoritmo de Dijkstra se usa solo cuando tiene una sola fuente y desea conocer la ruta más pequeña de un nodo a otro.

- El algoritmo Floyd-Warshall se usa cuando cualquiera de los nodos puede ser una fuente, por lo que desea que la distancia más corta llegue a cualquier nodo de destino desde cualquier nodo fuente. Esto solo falla cuando hay ciclos negativos.

- Bellman-Ford se usa como Dijkstra, cuando solo hay una fuente. Esto puede manejar pesos negativos.