“Año de la Universalización de la Salud”

FACULTAD DE INGENIERÍA DE PRODUCCIÓN Y SERVICIOS
ESCUELA PROFESIONAL DE INGENIERÍA DE SISTEMAS

UNIVERSIDAD NACIONAL DE SAN AGUSTÍN

ESTRUCTURAS DE DATOS Y ALGORITMOS

LABORATORIO 9

- **GUERRA ROSAS, RODRIGO RAUL**

DOCENTE : Edith Pamela Rivero Tupac

Arequipa - Perú

2021

Ejercicio 1.-
Se implementó en base al método visto en clase, usando clases como Vértice(Vertex), Arista(Edge), LinkedList como lista de adyacencia, Node como tipo de dato del LinkedList, y clase GraphLink.

Clase Vertice:

```java
public class Vertex<E> {
    protected E data;
    protected LinkedList<Edge<E>> listAdj;

    public Vertex(E data){
        this.data = data;
        listAdj = new LinkedList<Edge<E>>();
    }
    public E getData(){
        return this.data;
    }
    @Override
    public boolean equals(Object obj) {
        if(obj instanceof Vertex<?>){
            Vertex<E> v = (Vertex<E>) obj;
            return this.data.equals(v.data);
        }
        return false;
    }
    public String toString(){
        return this.data + "-->" + this.listAdj.toString()+"\n";
    }
}
```

Clase Arista:

```java
public class Edge<E> {
    protected Vertex<E> refDest;
    public int weight;

    public Edge(Vertex<E> refDest){
        this(refDest, -1);
    }
```

```
    public Edge(Vertex<E> refDest, int weight){
        this.refDest = refDest;
        this.weight = weight;
    }
    public boolean equals(Object obj){
        if(obj instanceof Edge<?>){
            Edge<E> e = (Edge<E>) obj;
            return this.refDest.equals(e.refDest);
        }
        return false;
    }
    public String toString(){
       if(this.weight > -1){
           return refDest.data+" ["+this.weight+"], ";
       }
       else{
           return refDest.data+", ";
       }
    }
}
```

Clase LinkedList:

```
public class LinkedList<T> {
    protected Node<T> first;

    public LinkedList(){
        this.first = null;
    }
    public boolean isEmpty(){
        return this.first == null;
    }
    public T search(T data){
        Node<T> aux = this.first;
        while(aux != null && !aux.data.equals(data)){
            aux = aux.getNext();
        }
        if(aux != null){
            return aux.getData();
        }
        return null;
```

```
    }
    public void insertFirst(T data){
        this.first = new Node<T>(data, this.first);
    }
    public String toString(){
        String str = "";
        Node<T> aux = this.first;
        while(aux != null){
            str += aux.getData();
            aux = aux.getNext();
        }
        return str;
    }

}
```

Clase GraphLink:

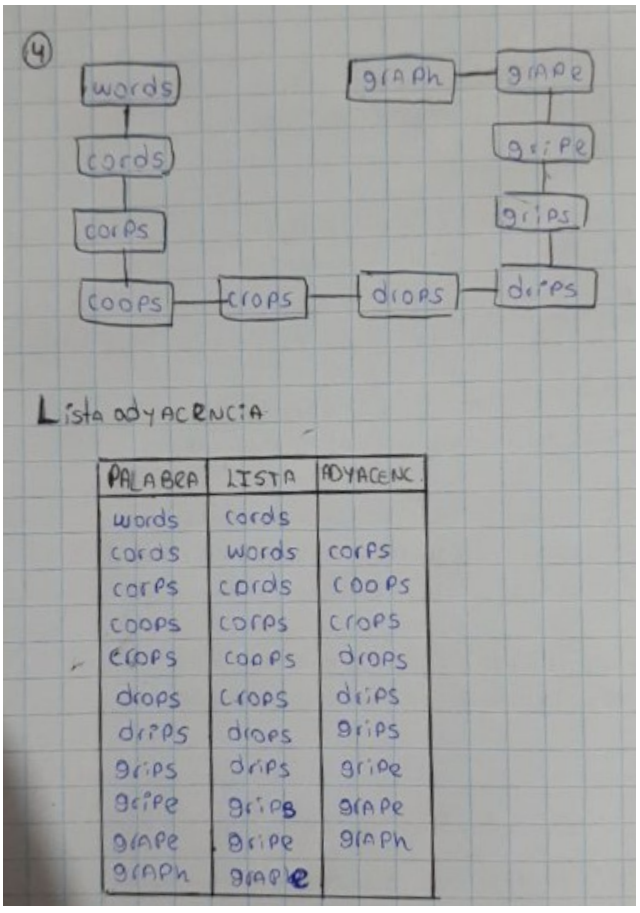
```
public class GraphLink<E> {
    protected LinkedList<Vertex<E>> vertexList;

    public GraphLink(){
        vertexList = new LinkedList<Vertex<E>>();
    }
    public void insertVertex(E data){
        Vertex<E> newVertex = new Vertex<E>(data);
        if(this.vertexList.search(newVertex) != null){
            System.out.println("Vertice ya ingresado");
            return;
        }
        this.vertexList.insertFirst(newVertex);
    }
    public void insertEdge(E verOrigin, E verDest){
        insertEdge(verOrigin, verDest, -1);
    }
    public void insertEdge(E verOrigin, E verDest, int weight){
        Vertex<E> refOrigin = this.vertexList.search(new
Vertex<E>(verOrigin));
        Vertex<E> refDest = this.vertexList.search(new
Vertex<E>(verDest));
        if(refOrigin == null || refDest == null){
```

```
            System.out.println("Vertice origen y/o destino no existen");
            return;
        }
        if(refOrigin.listAdj.search(new Edge<>(refDest)) != null){
            System.out.println("Arista insertada anteriormente");
            return;
        }
        refOrigin.listAdj.insertFirst(new Edge<E>(refDest,weight));
        refDest.listAdj.insertFirst(new Edge<E>(refOrigin,weight));
    }
    public String toString(){
        return this.vertexList.toString();
    }
}
```

EJERCICIO 2.-

EJERCICIO 3.-



EJERCICIO 4.-

CUESTIONARIO

¿Cuántas variantes del algoritmo de Dijkstra hay y cuál es la diferencia entre ellas?

Pues por lo que averigue y recuerdo de estructuras de datos hay muchas variantes que se basan y/o mezclan el algoritmo de Dijkstra para mejorar las fallas del mismo tales como que en grafo ponderado no soporte pesos negativos a diferencia de Bellman-Ford, tal como el Jhonson que mezcla estos dos antes mencionados. En general hay tantas variantes como gente que quiera crear nuevos algoritmos de busqueda de caminos en grafos.

Investigue sobre los ALGORITMOS DE CAMINOS MÍNIMOS e indique, ¿Qué similitudes encuentra, qué diferencias, en qué casos utilizar y porque?

Pues las similitudes son que la mayoría recorren y hacen comparaciones para hallar el menor peso, y asi recorriendo, diferencias en que algunos aceptan pesos negativos, unos son mejores que otros dependiendo de de la densidad del grafo, en casos de tener pequeños grafos conviene usar el algoritmo de Johnson y Floyd-Warshall siendo el primero mas rapido, en grafos muchos mas grandes conviene el Dijkstra y el Bellman-Ford, sinceramente prefiero Dijkstra salvo por lo de los pesos negativos seria mejor, tambien se podria utilizar el Johnson nuevamente para solucionar esto.