UNIVERSIDAD NACIONAL DE SAN AGUSTÍN DE AREQUIPA

FACULTAD DE INGENIERIA DE PRODUCCION Y SERVICIOS

ESCUELA PROFESIONAL DE INGENIERÍA DE SISTEMAS

Curso**: Estructura de datos**

Tema**: Grafos**

Elaborado Por:

- Anderson Rivera Quispe

Mayo 2021

Arequipa - Perú

```java
public class Node<Type> {
    protected Type data;
    protected Node<Type>next;

    public Node(Type data) {
        this.data=data;
        this.next=null;
    }
    public Node(Type data, Node<Type>next) {
        this.data=data;
        this.next=next;
    }
    public Type getInfo() {return data;}
    public Node<Type>getNext(){return next;}
    public void setData(Type data) {this.data=data;}
    public void setNext(Node<Type>next) {this.next=next;}

}
```

```java
public class ListLinked<T> {
    protected Node<T> first;

    public ListLinked() {
        this.first=null;
    }
    public Node<T>getFirst(){
        return first;
    }
    public void setFirst(Node<T> first) {
        this.first=first;
    }
    boolean isEmpty() {
        return this.first==null;
    }
    public T search(T data) {
        Node<T> nodo=this.first;
        while(nodo!= null && !nodo.data.equals(data))
            nodo=nodo.getNext();
        if(nodo != null)
            return nodo.data;
        return null;
    }
    void insertFirst(T data) {
        this.first=new Node<T>(data, this.first);
    }

    public String toString() {
        String r="";
        Node<T> aux = this.first;
        while(aux!=null) {
            r=r+aux.getInfo();
            aux=aux.getNext();
        }
        return r;
    }
}
```

```java

public class Vertex<E>{
    protected E data;
    protected ListLinked<Edge<E>> listAdj;
    protected int label;
    protected Vertex<E>path;

    public Vertex (E data) {
        this.data=data;
        listAdj=new ListLinked<Edge<E>>();
    }

    public E getData() {
        return data;
    }
    public boolean equals(Object o) {
        if(o instanceof Vertex<?>) {
            Vertex<E> v=(Vertex<E>)o;
            return this.data.equals(v.data);
        }
        return false;
    }

    public String toString() {
        return this.data+" --> "+this.listAdj.toString()+"\n";
    }

}

```

```java

public class Edge<E> {
    protected Vertex<E> refDest;
    protected 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 o) {
        if(o instanceof Edge<?>) {
            Edge<E> e = (Edge<E>)o;
            return this.refDest.equals(e.refDest);
        }
        return false;
    }

    public String toString() {
        if(this.weight > -1)return refDest.data+" ["+this.weight+"], ";
        else return refDest.data+", ";
    }

}

```

```java
public class Graph<E> {
    protected ListLinked<Vertex<E>> listVertex;

    public Graph() {
        listVertex= new ListLinked<Vertex<E>>();
    }

    public void insertVertex(E data) {
        Vertex<E>nuevo= new Vertex<E>(data);
        if(this.listVertex.search(nuevo)!=null) {
            System.out.println("Vertice ya fue insertado");
            return;
        }
        this.listVertex.insertFirst(nuevo);
    }
    public void insertEdge(E verOri, E verDes) {
        insertEdge(verOri,verDes, -1);
    }
    public void insertEdge(E verOri, E verDes,int weight) {
        Vertex<E> refOri= this.listVertex.search(new Vertex <E>(verOri));
        Vertex<E> refDes= this.listVertex.search(new Vertex <E>(verDes));
        if(refOri==null || refDes==null) {
            System.out.print("Vertice origen y/o destino no existen");
            return;
        }
        if(refOri.listAdj.search(new Edge<E>(refDes))!=null) {
            System.out.println("Arista ya fue insertada anteriormente");
            return;
        }
        refOri.listAdj.insertFirst(new Edge<E>(refDes,weight));
        //refDes.listAdj.insertFirst(new Edge<E>(refOri,weight));
    }
    public String toString() {
        return this.listVertex.toString();
    }
}
```

```java
public class Test {
    public static void main(String []args) {
    Graph<String> g= new Graph<String>();

    g.insertVertex("A");
    g.insertVertex("B");
    g.insertVertex("C");
    g.insertVertex("D");
    g.insertVertex("E");
    g.insertVertex("F");
    g.insertVertex("G");
    g.insertVertex("H");
    g.insertVertex("I");


    g.insertEdge("A", "B",18);
    g.insertEdge("B", "D",10);
    g.insertEdge("C", "E",8);
    g.insertEdge("G", "F",9);


    System.out.println(g);
    }


}
```

```
I -->
H -->
G --> F [9],
F -->
E -->
D -->
C --> E [8],
B --> D [10],
A --> B [18],
```

Grafo ponderado dirigido : Lista de adyacencia.

```java
public class Queue{
    private final int size = 20;
    private int[] queArray;
    private int front;
    private int rear;

    public Queue(){
        queArray = new int[size];
        front = 0;
        rear = -1;
    }

    public void insert(int i){
        if(rear == size-1)
            rear = -1;
        queArray[++rear] = i;
    }

    public int remove(){
        int temp = queArray[front++];
        if(front==size)
            front = 0;
        return temp;
    }

    public boolean isEmpty(){
        return (rear+1==front || front+size-1==rear);
    }
}
```

```java
public class Stack{
    private final int size = 20;
    private int[] st;
    private int top;

    public Stack(){
        st = new int[size];
        top = -1;
    }

    public void push(int i){
        st[++top] = i;
    }

    public int pop(){
        return st[top--];
    }

    public int peek(){
        return st[top];
    }

    public boolean isEmpty(){
        return (top==-1);
    }
}
```

```java
public class Vertex{
    public char label;
    public boolean wasVisited;

    public Vertex(char lab){
        label = lab;
        wasVisited = false;
    }
}
```

```java
public class Graph{
    private final int max_verts = 20;
    Vertex vertexList [];
    private int adjMat [] [];
    private int nVerts;
    private Stack theStack;
    private Queue theQueue;

    public Graph(){
        vertexList = new Vertex[max_verts];
        adjMat = new int [max_verts][max_verts];
        nVerts = 0;
        for(int j=0; j<max_verts; j++)
            for(int k=0;k<max_verts; k++)
                adjMat [j] [k] = 0;
        theStack = new Stack ();
    }

    public void addVertex(char lab){
        vertexList [nVerts ++] = new Vertex (lab);
    }

    public void addEdge(int start,int end){
        adjMat[start][end] = 1;
        adjMat[end][start] = 1;
    }

    public void displayVertex(int v){
        System.out.print(vertexList[v].label);
    }

    public void dfs(){
        vertexList [0] .wasVisited = true;
        displayVertex (0);
        theStack.push (0);

        while(!theStack.isEmpty()){
            int v = getAdjUnvisitedVertex_dfs (theStack.peek ());
            if(v == -1){

                theStack.pop();
            }else{

                vertexList[v].wasVisited = true;
                displayVertex(v);
                theStack.push(v);
            }
        }
    }

    public int getAdjUnvisitedVertex_dfs(int v){
        for (int j = 0; j <nVerts; j ++)
            if(adjMat[v][j]==1 && vertexList[j].wasVisited == false)
                return j;
        return -1; //
    }
```

```
    public void bfs(){
        vertexList[0].wasVisited = true;
        displayVertex(0);
        theQueue.insert(0);
        int v2;

        while(!theQueue.isEmpty()){
            int v1 = theQueue.remove();
            while((v2 = getAdjUnvisitedVertex_bfs(v1))!=-1){
                vertexList[v2].wasVisited = true;
                displayVertex(v2);
                theQueue.insert(v2);
            }
        }

        for(int j=0;j<nVerts;j++)
            vertexList[j].wasVisited = false;
    }

    public int getAdjUnvisitedVertex_bfs(int v){
        for(int j=0;j<nVerts;j++)
            if(adjMat[v][j]==1&&vertexList[j].wasVisited == false)
                return j;
        return -1;
    }

}
```

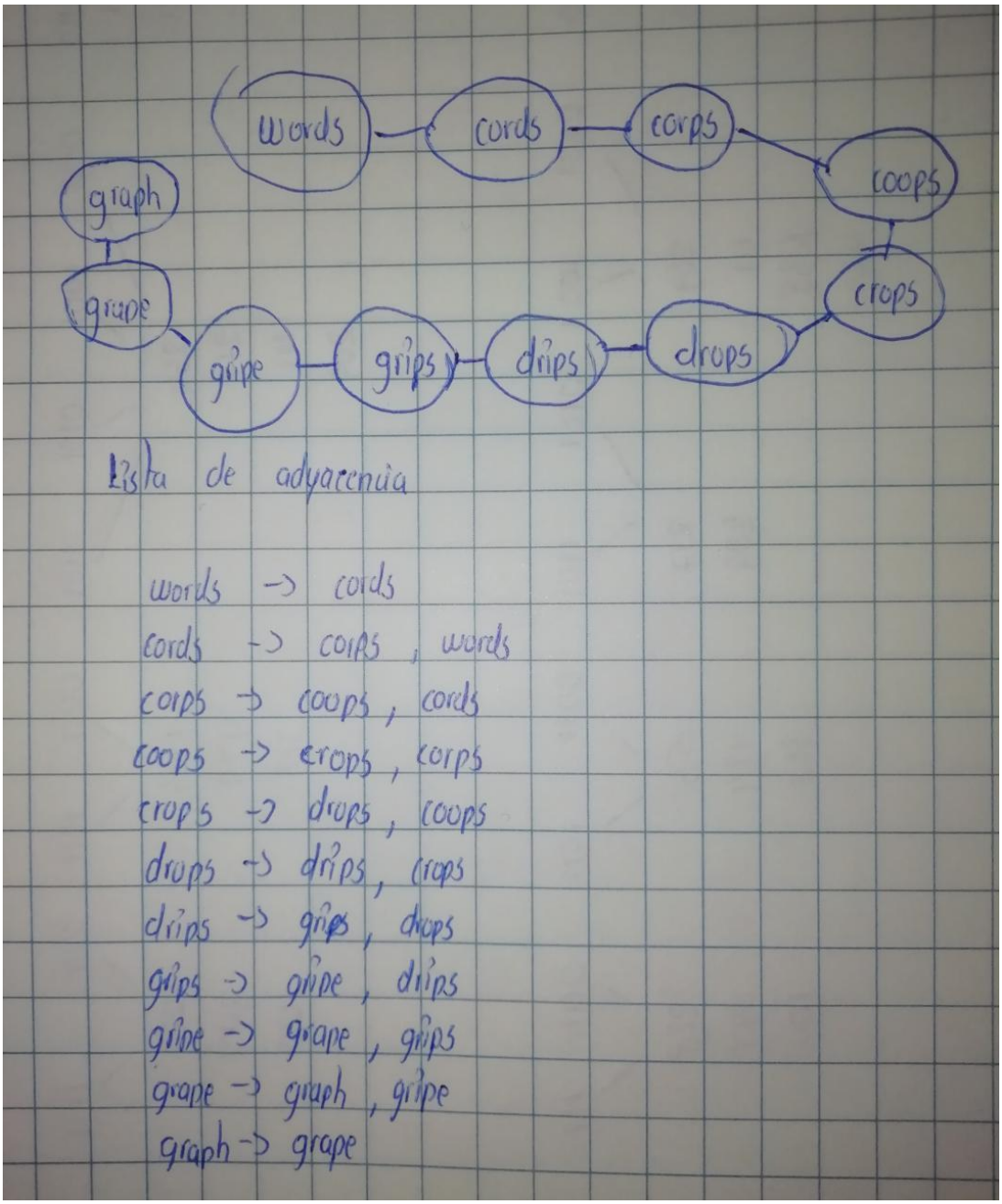
```
import java.util.Scanner;

public class Test{
    public static void main(String[] args){
        Graph theGraph = new Graph();
        theGraph.addVertex('A');
        theGraph.addVertex('B');
        theGraph.addVertex('C');
        theGraph.addVertex('D');
        theGraph.addVertex('E');

        theGraph.addEdge(0, 1);
        theGraph.addEdge(1, 2);
        theGraph.addEdge(0, 3);
        theGraph.addEdge(3, 4);

        System.out.print("Visits: ");
        theGraph.dfs();
        System.out.println();
        theGraph.bfs();
    }
}
```

4.


words
cords
corps
coops
crops
drops
drips
grips
gripe
grape
graph
Lista de adyacencia
words -> cords
cords -> corps , words
corps -> coops , cords
coops -> crops , corps
crops -> drops , coops
drops -> drips , crops
drips -> grips , drops
grips -> gripe , drips
gripe -> grape , grips
grape -> graph , gripe
graph -> grape