

Estructura de datos avanzadas

GRAFOS



Por: Ing. Edwin Calle Terrazas

Un grafo se define como un conjunto de vértices (nodos) y un conjunto de arcos (aristas)

GRAFOS NO DIRIGIDOS: Arcos (no orientados).

$$(u,v) = (v,u)$$

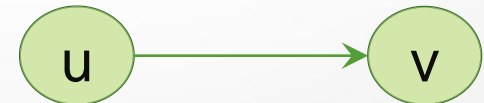


Grado de un vértice:

Número de arcos que lo contienen

GRAFOS DIRIGIDOS: Arcos (con dirección).

$$(u,v) \neq (v,u)$$



Grado de salida de un vértice u:

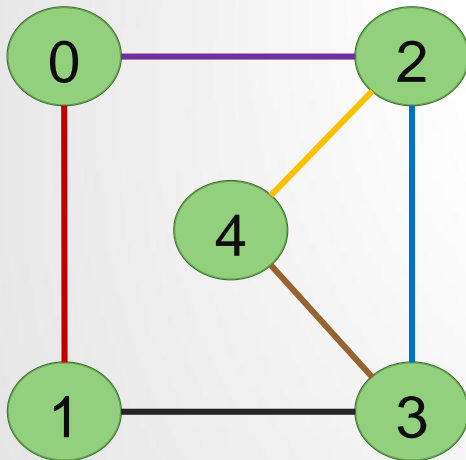
Número de arcos cuyo vértice inicial es u

Grado de entrada de un vértice u:

Número de arcos cuyo vértice final es u

REPRESENTACIÓN DE LOS GRAFOS

REPRESENTACIÓN MEDIANTE MATRICES DE ADYACENCIA



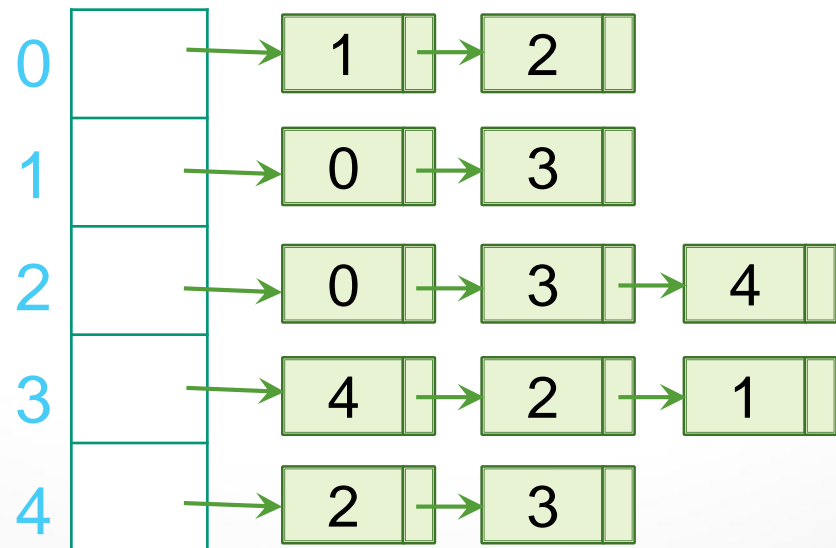
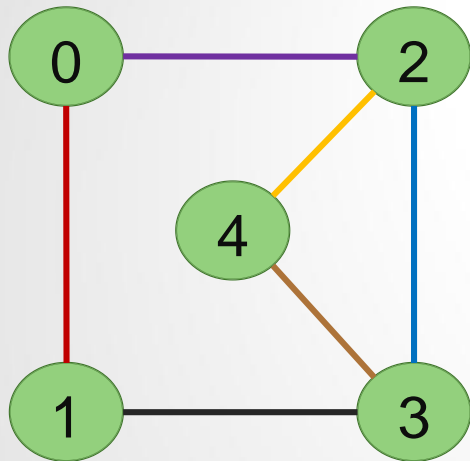
	0	1	2	3	4
0	0	1	1	0	0
1	1	0	0	1	0
2	1	0	0	1	1
3	0	1	1	0	1
4	0	0	1	1	0

Matriz de adyacencia

$M[u][v] = 1$, Existe un arco de $u \rightarrow v$ & $v \rightarrow u$

$M[u][v] = 0$, No existe arco

REPRESENTACIÓN MEDIANTE LISTAS DE ADYACENCIA



RECORRIDOS SOBRE GRAFOS

Se parte de un nodo dado y se visitan los vértices del grafo de manera ordenada y sistemática, pasando de un vértice a otro a través de los arcos del grafo.

BÚSQUEDA PRIMERO EN PROFUNDIDAD DFS (Depth-First Search)

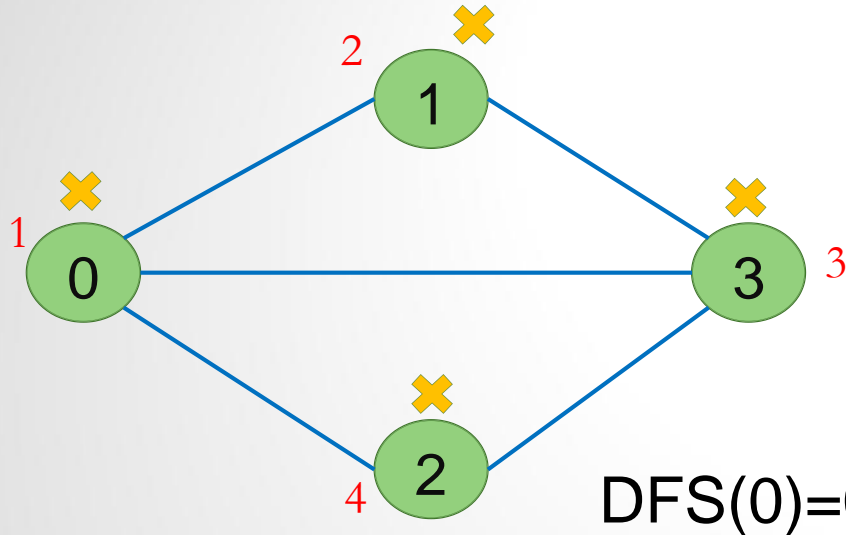
Equivalente al recorrido en preOrden de un árbol

BÚSQUEDA PRIMERO EN ANCHURA BFS (Breadth-First Search)

Equivalente al recorrido por niveles de un árbol

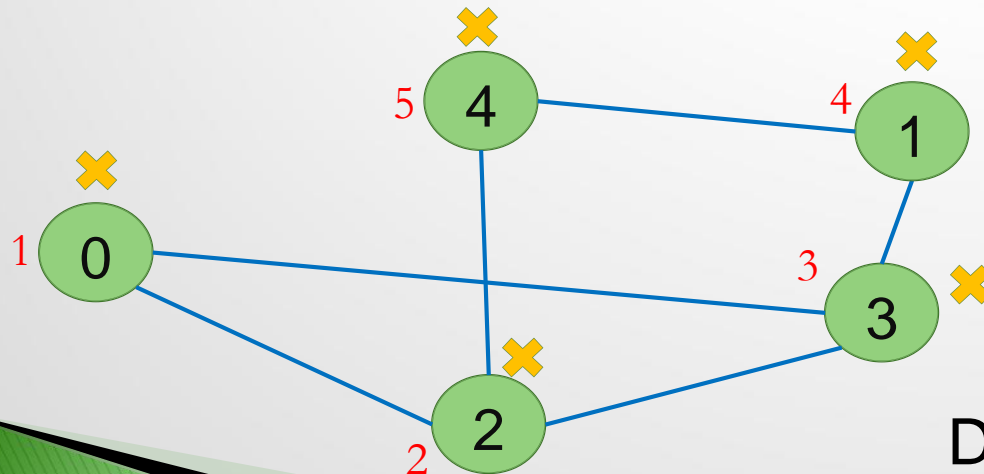


RECORRIDO DFS

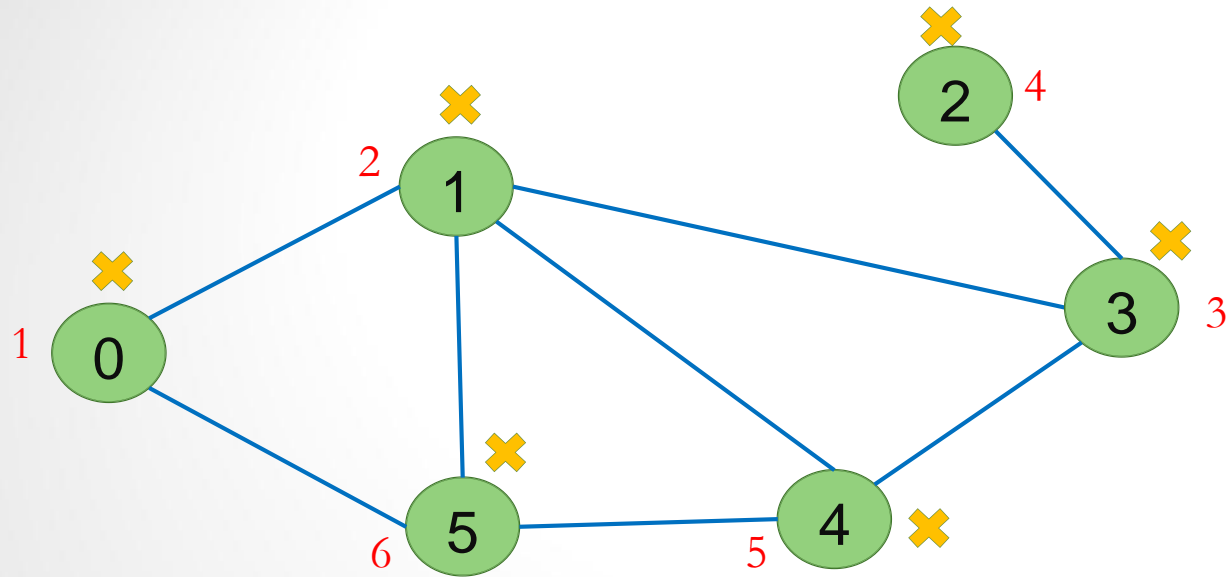


Empezamos marcando el vértice de inicio (v), luego recorremos buscando todos sus adyacentes de v que NO han sido marcados (los muestra y los marca)

Si ya no encuentra mas adyacentes que no están marcados, vuelve por su predecesor y si encuentra algún adyacente lo muestra y lo marca.



DFS(0)=0 2 3 1 4

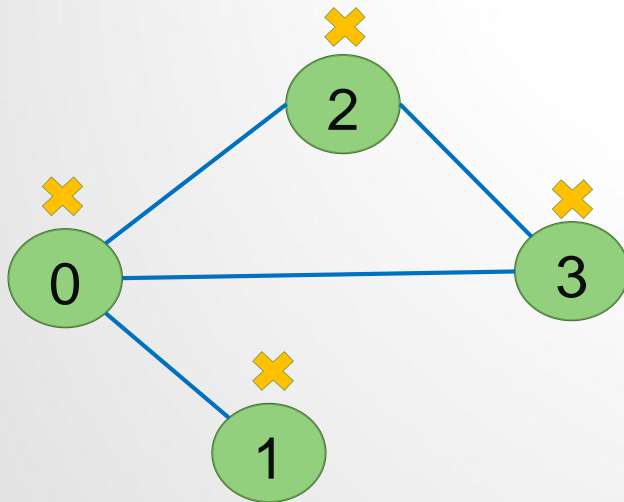


DFS(0)=0 1 3 2 4 5

RECORRIDO BFS

El algoritmo no tiene naturaleza recursiva. Utilizaremos una cola (C). Se toma en cuenta:

1. Todo lo que se introduce a la cola se marca
2. Todo lo que sale de la cola se visita (imprime)



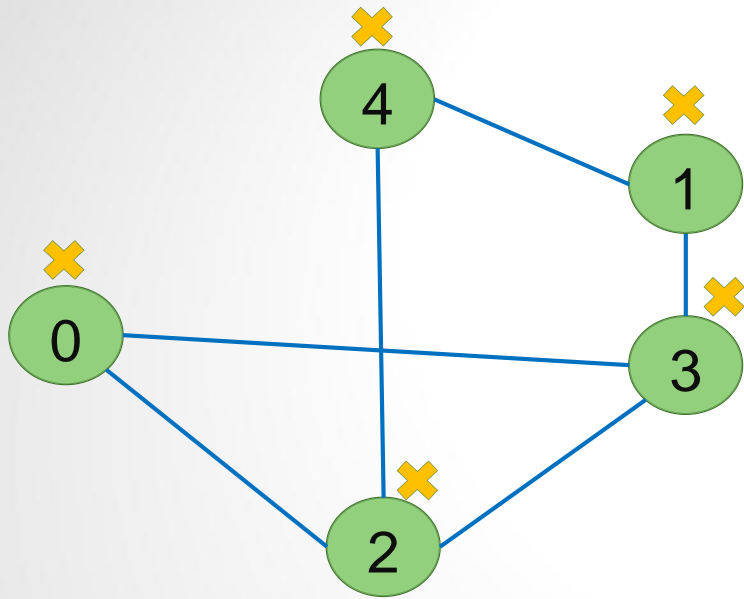
Empezamos insertando a la cola el vértice de inicio (v) y lo marcamos.

Dentro del ciclo, sacamos el elemento de la Cola y luego lo muestra.

Luego todos los adyacentes de v que no han sido visitados pasan a la Cola, luego repetimos al ciclo hasta que la cola queda vacía.



BFS(2)= 2 0 3 1



BFS(4)=4 1 2 3 0

IMPLEMENTACIÓN A TRAVÉS DE MATRIZ DE ADYACENCIA

```
public class Grafo {  
    private static int max=50;  
    private int M[][];  
    private int n;  
  
    public Grafo(){  
        M = new int[max][max];  
        for(int i =0; i<max; i++){  
            for(int j=0; j<max; j++){  
                M[i][j] = 0;  
            }  
        }  
        n = -1;  
        marca = new boolean[max+1];  
    }  
}
```

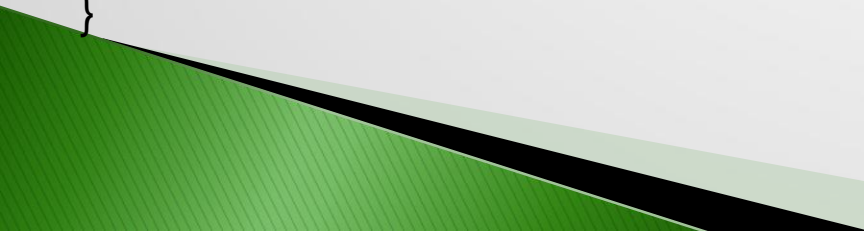
```
public void crearVertice(){
    if (n == max){
        JOptionPane.showMessageDialog(null, "Numero de vertice igual a" + max+1);
        return;
    }
    n++;
}

public int cantVertices() { return n+1; }

public boolean esVerticeValido(int v){
    return (v>=0 && v<=n);
}

public void insertarArco(int u, int v){
    if (!esVerticeValido(u) || !esVerticeValido(v)){
        JOptionPane.showMessageDialog(null, "No es un vertice valido"); return ;
    }
    M[u][v] = 1;
    M[v][u] = 1;
}

public void eliminarArco(int u, int v){
    if (!esVerticeValido(u) || !esVerticeValido(v))
        return; //No existe el vertice u o el vertice v.
    M[u][v] = 0;
    M[v][u] = 0;
}
```



```
private boolean marca[];
```

```
private void desmarcarTodos(){  
    for (int i = 0; i <= n; i++) {  
        marca[i] = false;  
    }  
}
```

```
private void marcar(int u){  
    if (esVerticeValido(u))  
        marca[u] = true;  
}
```

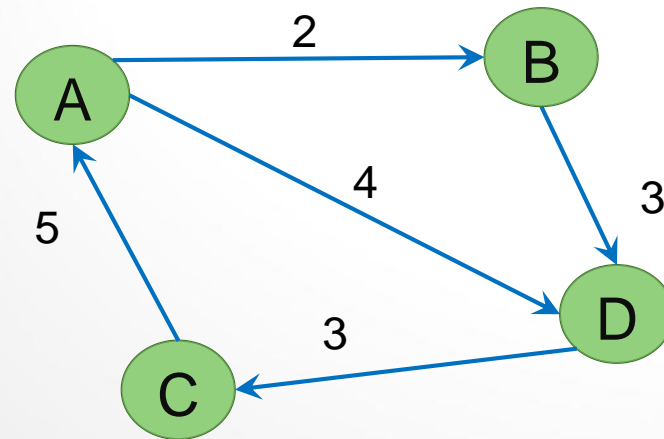
```
private void desmarcar(int u){  
    if (esVerticeValido(u))  
        marca[u] = false;  
}
```

```
private boolean esMarcado(int u){ //Devuelve true, si el vertice u está marcado.  
    return marca[u];  
}
```

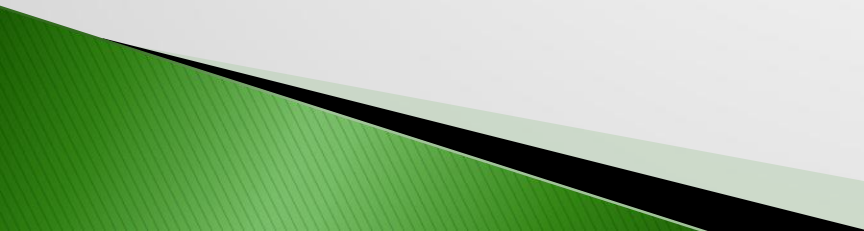
```
public int getArco(int i, int j){  
    return M[i][j];  
}
```



IMPLEMENTACIÓN A TRAVÉS DE LISTAS DE ADYACENCIA (Listas enlazadas)



```
public class Arco
{
    private Vertice verticeD;
    private float costo;
    private Arco prox;
    public Arco(Vertice verD,float precio, Arco proxA) {
        verticeD=verD;
        costo=precio;
        prox=proxA;
    }
    public void setCosto(float c){    costo=c;    }
    public float getCosto(){    return costo;    }
    public void setVerticeD(Vertice ptr){    verticeD=ptr;    }
    public Vertice getVerticeD(){    return verticeD;    }
    public void setProx(Arco p){    prox=p;    }
    public Arco getProx(){    return prox;    }
}
```



```
public class Vertice {  
    private Vertice prox;  
    private String nombre;  
    private Arco primeroA;
```

```
    public Vertice(Vertice proxV,String nomb,Arco priA){  
        prox=proxV;  
        nombre=nomb;  
        primeroA=priA;  
    }
```

```
    public void setNombre(String nom){    nombre=nom;    }
```

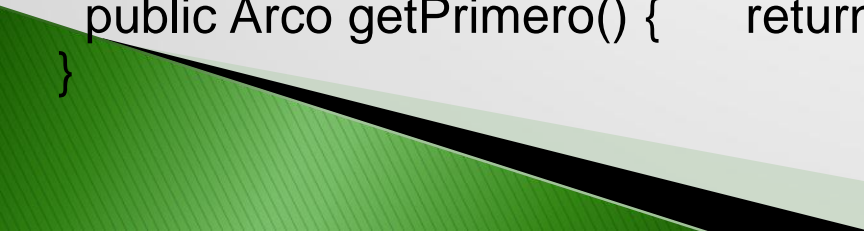
```
    public String getNombre(){    return nombre;    }
```

```
    public void setProx(Vertice proxV){    prox=proxV;    }
```

```
    public Vertice getProx(){    return prox;    }
```

```
    public void setPrimero(Arco priA) {    primeroA=priA;    }
```

```
    public Arco getPrimero() {    return primeroA;    }  
}
```



```
public class Grafo {
    private Vertice primeroV;

    public Grafo() {
        primeroV=null;
    }
    public Vertice crearVertice(Vertice proxV, String nomb, Arco priA){
        Vertice p=new Vertice(proxV,nomb,priA);
        if(p==null)
            JOptionPane.showMessageDialog(null, "No existe espacio de memoria");
        return p;
    }
    public Arco crearArco(Vertice verD, float precio,Arco proxA){
        Arco q=new Arco(verD,precio,proxA);
        if(q==null)
            JOptionPane.showMessageDialog(null, "No existe espacio de memoria");
        return q;
    }
    public Vertice buscarVertice(String nom) {
        Vertice p=primeroV;
        while( p!=null){
            if(p.getNombre().equals(nom))
                return p;
            else
                p=p.getProx();
        }
        return null;
    }
}
```




```
public void insertarArco(String A, String B, float cost){  
    Vertice pa=buscarVertice(A);  
    Vertice pb=buscarVertice(B);  
    if(pa==null)  
        pa=primeroV=crearVertice(primeroV,A,null);  
    if(pb==null)  
        pb=primeroV=crearVertice(primeroV,B,null);  
  
    pa.setPrimero(crearArco(pb,cost,pa.getPrimero()));  
}
```

```
public void imprimir(JTextArea ta)  
{  
    Vertice p=primeroV;  
    Arco q;  
    while( p!=null ){  
        q=p.getPrimero();  
        while( q!=null ){  
            ta.append(p.getNombre() + " --> " + q.getVerticeD().getNombre() + " " +  
                q.getCosto());  
            ta.append("\n");  
            q=q.getProx();  
        }  
        p=p.getProx();  
    }  
}  
}  
} //end class
```

IMPLEMENTACIÓN A TRAVÉS DE LISTAS DE OBJETOS

```
public class Arco {  
    private float costo;  
    private Vertice verticeD;  
  
    public Arco(Vertice vd, float co) {  
        this.costo = co;  
        this.verticeD = vd;  
    }  
  
    public void setCosto(float co) {  
        this.costo = co;  
    }  
  
    public float getCosto() {  
        return costo;  
    }  
  
    public String getNombreVertD() {  
        return verticeD.getNombre(); //Devuelve el nombre del vertice destino  
    }  
  
    public void setNombreVertD(Vertice vd) {  
        this.verticeD = vd;  
    }  
}
```



```
public class Vertice
{
    String nombre;
    public Lista LArcos;

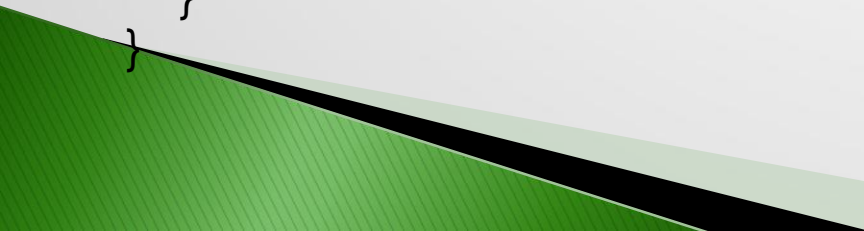
    public Vertice(String nom) {
        this.nombre = nom;
        this.LArcos = new Lista();
    }

    public void setNombre(String nom) {
        this.nombre = nom;
    }

    public String getNombre() {
        return this.nombre;
    }

    public int getCantArcos() {
        return LArcos.dim();
    }

    public void insertarArco(Arco arco) {
        LArcos.insertarUlt(arco); //Inserta el arco q ya viene creado
    }
}
```

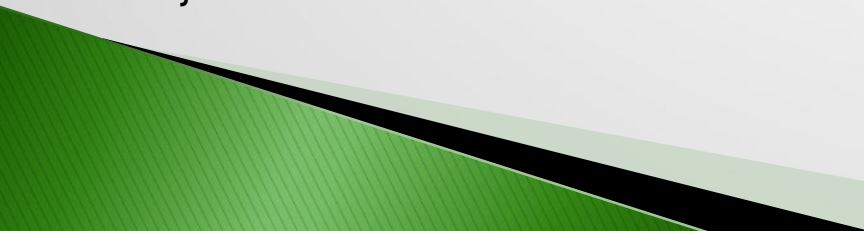


```
public class Grafo
{
    public Lista LVertices;

    public Grafo() {
        LVertices = new Lista();
    }

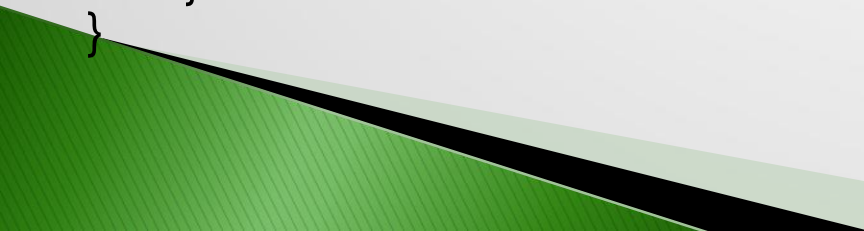
    public void crearVertice(String nomV){
        LVertices.insertarUlt(new Vertice(nomV));
    }

    public Vertice buscarVertice(String nomV)
    {
        Vertice vertice;
        int i=0;
        while (i<LVertices.dim())
        {
            vertice =(Vertice)LVertices.getElem(i);
            if (vertice.getNombre().equals(nomV))
                return vertice;
            i++;
        }
        return null;
    }
}
```



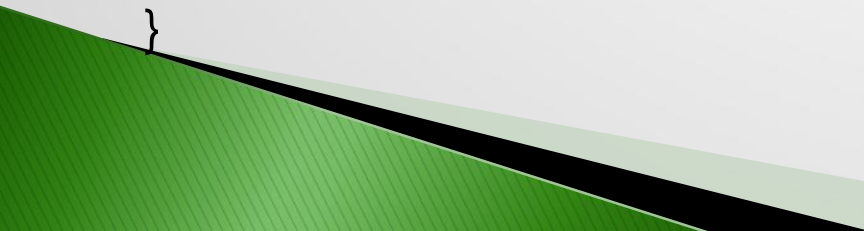
```
public void insertarArco(String X,String Y, float co) {  
    Vertice vo = buscarVertice(X);  
    Vertice vd = buscarVertice(Y);  
    vo.insertarArco(new Arco(vd, co));  
}
```

```
public void imprimir(JTextArea jta){  
    int i = 0,j; Vertice v; Arco a;  
    while (i < LVertices.dim())  
    {  
        v = (Vertice)LVertices.getElem(i);  
        j=0;  
        while (j<v.LArcos.dim()) {  
            jta.append(v.getNombre());  
            jta.append("-->");  
            a = (Arco)v.LArcos.getElem(j); //Muestra el arco donde apunto  
            jta.append(a.getNombreVertD() + " " + a.getCosto());  
            jta.append("\n");  
            j++;  
        }  
        i++;  
    }  
}
```



```
public void DFS(String A, JTextArea jta){  
    jta.append("DFS: ");  
    desmarcarTodos();  
    ordenarVerticesAlf();  
    Vertice a = buscarVertice(A);  
    dfs(a, jta);  
    jta.append("\n");  
}
```

```
private void dfs(Vertice v, JTextArea jta){  
    jta.append(v.getNombre() + " ");  
    v.marcado=true;  
    Arco a;  
    for (int i = 0; i < v.LArcos.dim(); i++) {  
        a = (Arco) v.LArcos.getElem(i);  
        Vertice w = buscarVertice(a.getNombreVertD());  
        if(!w.marcado)  
            dfs(w, jta);  
    }  
}
```



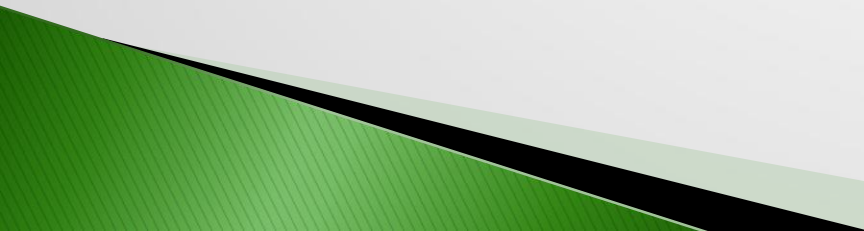
```

public void BFS(String s, JTextArea jta) {
    desmarcarTodos();
    ordenarVerticesAlf();  Arco a;
    Vertice v = buscarVertice(s), w;
    LinkedList <Vertice> C;
    C=new LinkedList<Vertice>();
    C.add(v);    v.marcado=true;
    jta.append("BFS: ");
    do{
        v = C.pop();
        jta.append(v.getNombre() + " ");
        for (int i = 0; i < v.LArcos.dim(); i++) {
            a = (Arco) v.LArcos.getElem(i);
            w = buscarVertice(a.getNombreVertD());
            if (!w.marcado) {
                C.add(w);
                w.marcado=true;
            }
        }
    }while (!C.isEmpty());
    jta.append("\n");
}

```

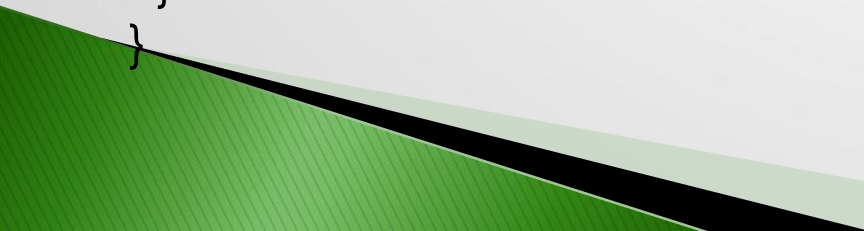
ORDENAR ALFABETICAMENTE POR NOMBRE DE VERTICES DESTINO DE CADA ARCO

```
public void ordenarArcosAlf() {  
    Arco aux; Arco a1; Arco a2;  
    for(int i=0;i<LArcos.dim();i++){  
        for(int j=0;j<LArcos.dim()-1;j++){  
            a1=(Arco)LArcos.getElem(j);  
            a2=(Arco)LArcos.getElem(j+1);  
            if(a1.getNombreVertD().compareTo(a2.getNombreVertD())>0){  
                aux=(Arco)LArcos.getElem(j);  
                LArcos.setElem(a2, j);  
                LArcos.setElem(aux, j+1);  
            }  
        }  
    }  
}
```



ORDENAR ALFABETICAMENTE POR NOMBRE DE LOS VERTICES

```
public void ordenarVerticesAlf() {  
    Vertice aux=new Vertice(); Vertice v1; Vertice v2;  
    for(int i=0;i<LVertices.dim();i++){  
        for(int j=0;j<LVertices.dim()-1;j++){  
            v1=(Vertice)LVertices.getElem(j);  
            v2=(Vertice)LVertices.getElem(j+1);  
            if(v1.getNombre().compareTo(v2.getNombre())>0){  
                aux=(Vertice)LVertices.getElem(j);  
                LVertices.setElem(v2, j);  
                LVertices.setElem(aux, j+1);  
            }  
        }  
    }  
    for(int i=0;i<LVertices.dim();i++){  
        Vertice v=(Vertice)LVertices.getElem(i);  
        v.ordenarArcosAlf();  
    }  
}
```



LISTA DE OBJETOS

```
public class Lista
{
    private static int max=30;
    private Object[] v;
    private int n;

    public Lista() {
        v = new Object[max];
        n = 0;
    }

    public void dim(int d){
        n = d;
    }

    public int dim(){
        return n;
    }

    public void setElem(Object x, int pos){
        v[pos] = x;
    }

    public Object getElem(int pos){
        return v[pos];
    }

    boolean vacia() {
        return n == 0;
    }

    boolean llena() {
        return n == max;
    }
}
```

```

public void insertar(Object x, int p){
    if (llenar())
        JOptionPane.showMessageDialog(null, "Lista::Lista llena...!!!");
    if (p < 0 || p > n)
        JOptionPane.showMessageDialog(null, "Lista::Posicion no valida...!!!!");
    else
    {
        int m = n - 1;
        while (m >= p){
            v[m + 1] = v[m];
            m = m - 1;
        }
        v[p] = x;
        n++;
    }
}

public void insertarPri(Object x) { insertar(x, 0); }

public void insertarUlt(Object x) { insertar(x, n); }

public void eliminar(int pos) {
    if (vaciar())
        JOptionPane.showMessageDialog(null, "Lista::Lista vacia...!!!");

    int k = pos + 1;
    while (k < n) {
        v[k - 1] = v[k];
        k = k + 1;
    }
    n = n - 1;
}

public void eliminarPri() { eliminar(0); }

public void eliminarUlt() { eliminar(n - 1); }

} //end class

```