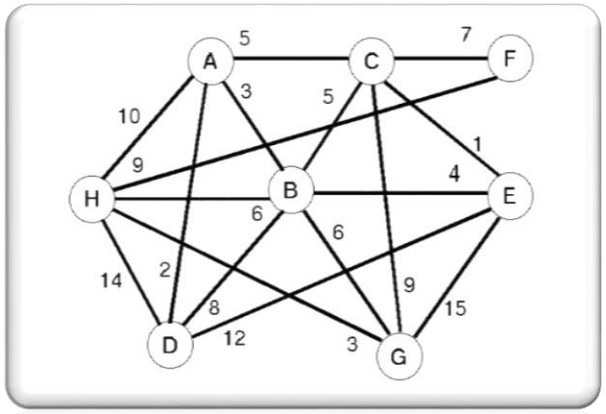
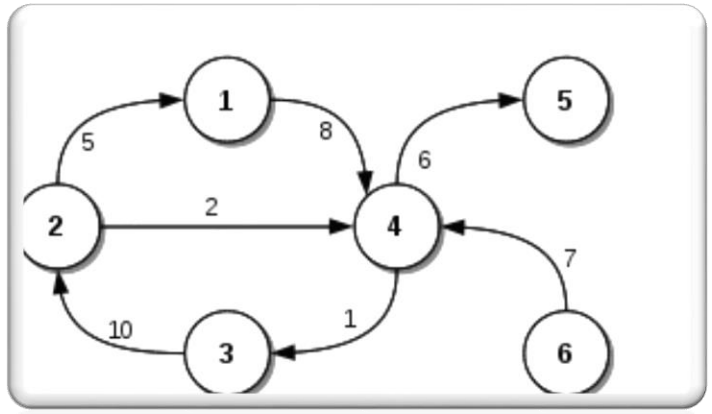
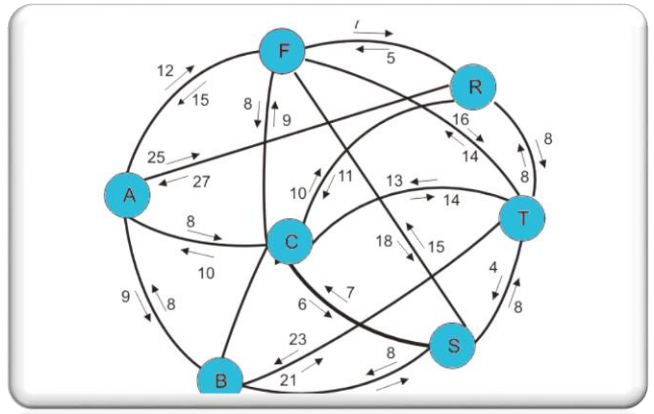
[Estructuras de Datos]







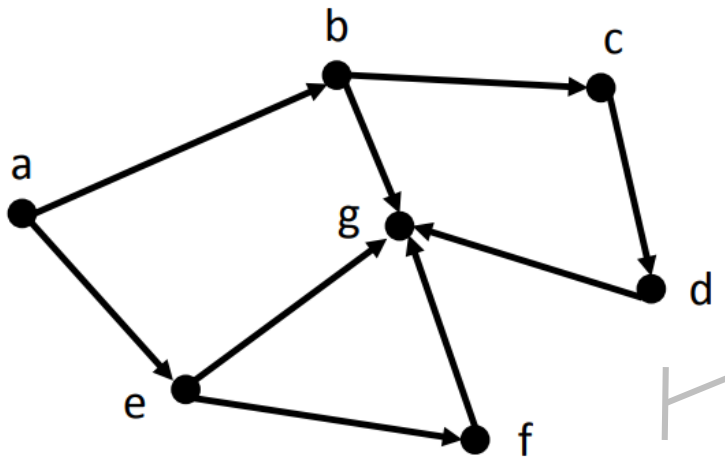
GRAFOS DIRIGIDOS Y NO DIRIGIDOS.

Copyright

GRAFOS DIRIGIDOS
VS.
GRAFOS NO DIRIGIDOS

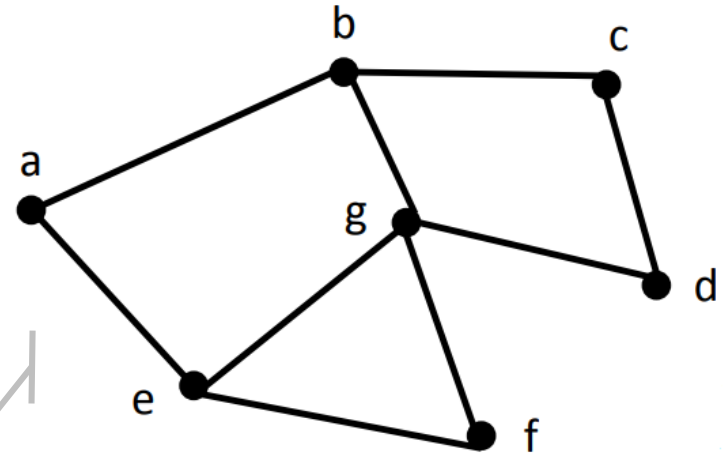
Introducción: grafos dirigidos vs. no dirigidos

- Un **Grafo** es una ED compuesta por un **conjunto de vértices V**, y un **conjunto de arcos E**.
- Un **arco** es un **par de vértices** que modela una relación entre estos vértices.
  - Cuando el par es **ordenado**, el grafo es **dirigido**.
  - Cuando el par es **desordenado**, el grado es **no dirigido**.



*El **arco (a,b)** representa que **desde** el vértice **a** se puede **llegar** al vértice **b**.*
*El **arco (b,a)** no existe.*

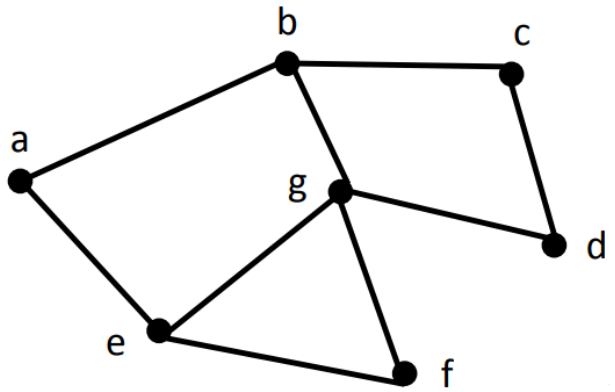
*Los **arcos (a,b)** y **(b,a)** representan lo mismo; existe una conexión **desde** el vértice **a** hacia el vértice **b** y viceversa.*



TDA Grafo :: Dirigido vs. No dirigido

TDA Grafo No Dirigido

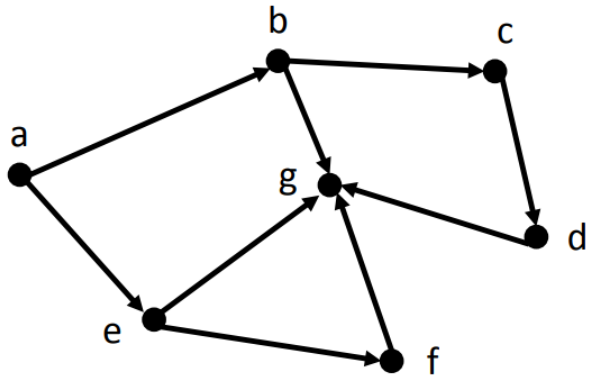
```
public interface Graph<V,E> {
  public Iterable<Vertex<V>> vertices();
  public Iterable<Edge<E>> edges();
  public Iterable<Edge<E>> incidentEdges(Vertex<V> v) throws InvalidVertexException;
  public Vertex<V> opposite(Vertex<V> v, Edge<E> e) throws InvalidVertexException, InvalidEdgeException;
  public Vertex<V> [] endvertices(Edge<E> e) throws InvalidEdgeException;
  public boolean areAdjacent(Vertex<V> v,Vertex<V> w) throws InvalidVertexException;
  public V replace(Vertex<V> v, V x) throws InvalidVertexException;
  public Vertex<V> insertVertex(V x);
  public Edge<E> insertEdge(Vertex<V> v, Vertex<V> w, E e) throws InvalidVertexException;
  public V removeVertex(Vertex<V> v) throws InvalidVertexException;
  public E removeEdge(Edge<E> e) throws InvalidEdgeException;
}
```




```
Vertices() → {a,b,c,d,e,f,g}
Edges() → {(a,b),(b,c),(c,d),(d,g),(g,b),(g,f),(f,e),(e,a),(e,g)}
IncidentEdges(a) → {(e,a), (a,b)}
IncidentEdges(g) → {(d,g),(g,b),(g,f),(e,g)}
Opposite(a, (a,b)) → b
Opposite(a, (e,a)) → e
areAdyacent(a,b) → true
areAdyacent(b,a) → true
```

TDA Grafo Dirigido

```
public interface GraphD<V,E> {
  public Iterable<Vertex<V>> vertices();
  public Iterable<Edge<E>> edges();
  public Iterable<Edge<E>> incidentEdges(Vertex<V> v) throws InvalidVertexException;
  public Iterable<Edge<E>> succesorEdges(Vertex<V> v) throws InvalidVertexException;
  public Vertex<V> opposite(Vertex<V> v, Edge<E> e) throws InvalidVertexException, InvalidEdgeException;
  public Vertex<V> [] endvertices(Edge<E> e) throws InvalidEdgeException;
  public boolean areAdjacent(Vertex<V> v,Vertex<V> w) throws InvalidVertexException;
  public V replace(Vertex<V> v, V x) throws InvalidVertexException;
  public Vertex<V> insertVertex(V x);
  public Edge<E> insertEdge(Vertex<V> v, Vertex<V> w, E e) throws InvalidVertexException;
  public V removeVertex(Vertex<V> v) throws InvalidVertexException;
  public E removeEdge(Edge<E> e) throws InvalidEdgeException;
}
```




```
Vertices() → {a,b,c,d,e,f,g}
Edges() → {(a,b),(b,c),(c,d),(d,g),(b,g),(f,g),(e,g),(e,f),(f,g)}
IncidentEdges(a) → {}
IncidentEdges(g) → {(b,g),(d,g),(f,g),(e,g)}
SuccesorEdges(e) → {(e,g),(e,f)}
Opposite(a, (a,b)) → b
Opposite(e, (a,e)) → Error
areAdyacent(a,b) → true
areAdyacent(b,a) → false
```

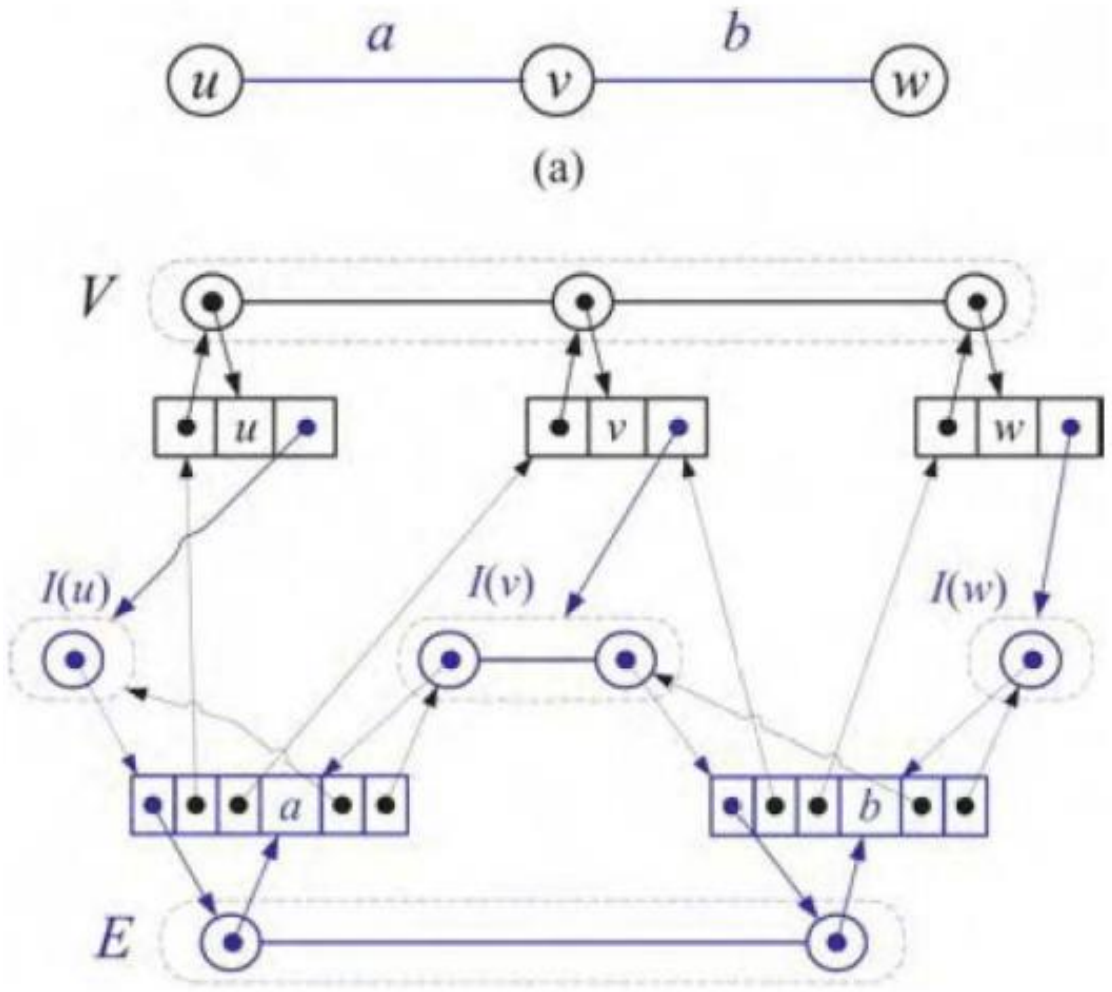
GRAFOS :: IMPLEMENTACIÓN CON LISTA DE ADYACENCIA

Grafo con lista de adyacencia

```
public class Vertice<V,E> implements Vertex<V>{
  private V rotulo;
  private PositionList<Arco<V,E>> adyacentes;
  private Position<Vertice<V,E>> posicionEnNodos;
  //Constructor, Seters y getter.
}
```

```
public class Arco<V,E> implements Edge<E>{
  private E rotulo;
  private Vertice<V,E> vertice1, vertice2; //No Dirigido
  private Vertice<V,E> predecesor, sucesor; //Dirigido
  private Position<Arco<V,E>> posAdyV1, posAdyV2;
  //Constructor, seters y getters.
}
```

```
public class Digrafo<V,E> implements GraphD<V,E>{
  protected PositionList<Vertice<V,E>> nodos;
  //Constructor y operaciones
}
public class Grafo<V,E> implements Graph<V,E>{
  protected PositionList<Vertice<V,E>> nodos;
  //Constructor y operaciones
}
```



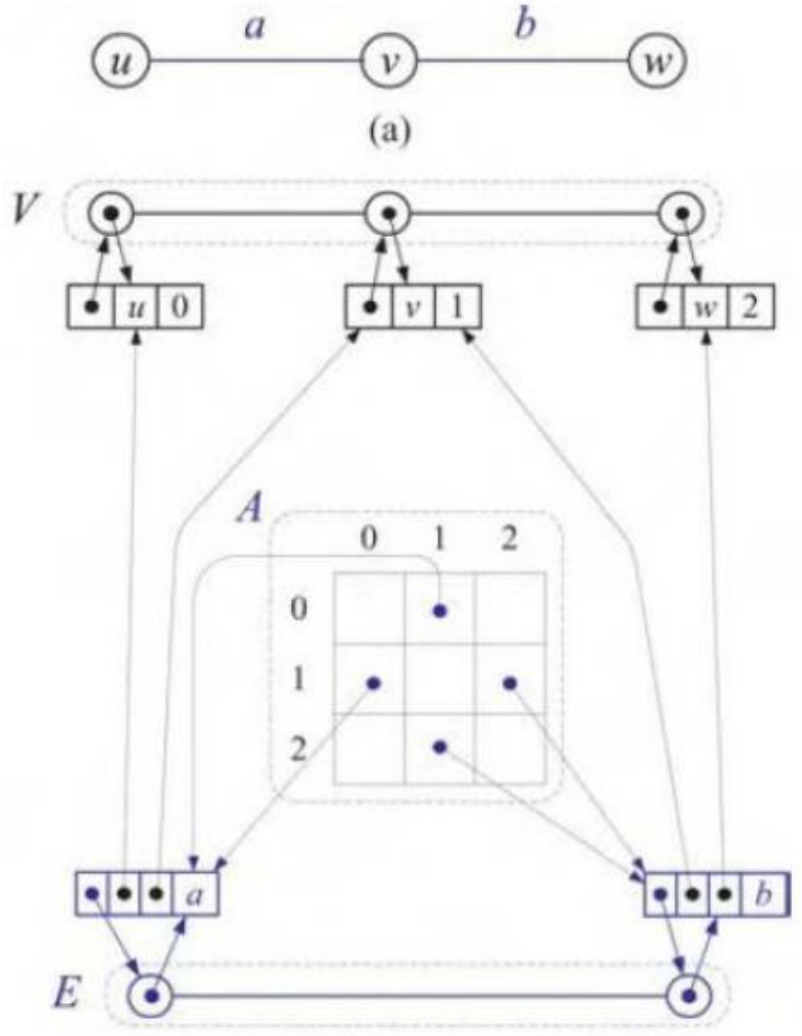
GRAFOS :: IMPLEMENTACIÓN CON MATRIZ DE ADYACENCIA

Grafo con matriz de adyacencia

```java
public class Vertice<V> implements Vertex<V> {
  private V rotulo;
  private int indice;
  private Position<Vertex<V>> posicionEnVertices;
  //Constructor, setter y getters.
}
```

```java
public class Arco<V,E> implements Edge<E>{
  private E rotulo;
  private Vertice<V,E> vertice1, vertice2; //No Dirigido
  private Vertice<V,E> predecesor, sucesor; //Dirigido
  private Position<Edge<E>> posicionEnArcos;
  //Constructor, seters y getters.
}
```

```java
//public class Grafo<V,E> implements Graph<V,E>{
//public class Digrafo<V,E> implements GraphD<V,E>{
  protected PositionList<Vertex<V>> vertices;
  protected PositionList<Edge<E>> arcos;
  protected Edge<E> [][] matriz;
  protected int cantidadVertices;
  //Constructor y operaciones
}
```



Fin de la presentación.