Algoritmos
y
Estructuras de Datos

Cursada 2015

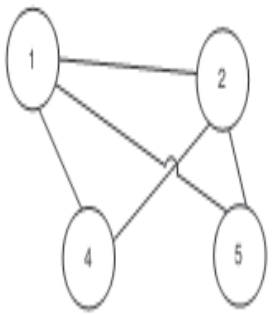
Grafos

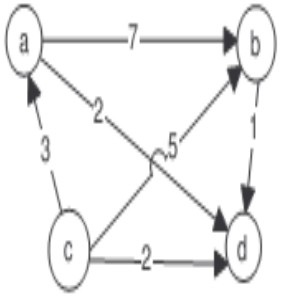
En general, un GRAFO se utiliza para representar relaciones arbitrarias entre entidades del mismo tipo. Las entidades reciben el nombre de NODOS o VÉRTICES y las relaciones entre ellos se denominan ARISTAS.

Los grafos son muy útiles en problemas que involucran estructuras tales como:

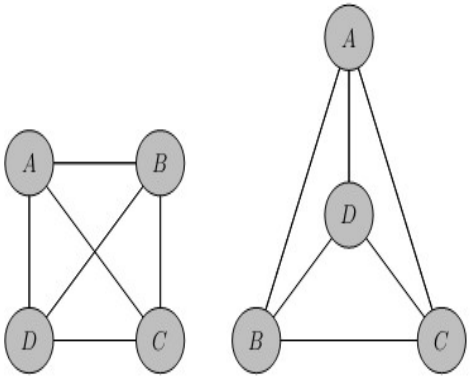
- **Circuitos Eléctricos,**
- **Tuberías de Agua**
- **Redes de Computadoras**
- **Redes Sociales**
- **Rutas Aéreas, Terrestres, Marítimas**
- **Etc**

Grafo SIN Peso – Grafo dirigido y con Peso





Dos Dibujos de un mismo Grafo



Grafos - Implementación

Todas las implementaciones de grafos deberían permitir operaciones como agregar o eliminar vértices, agregar o eliminar aristas.

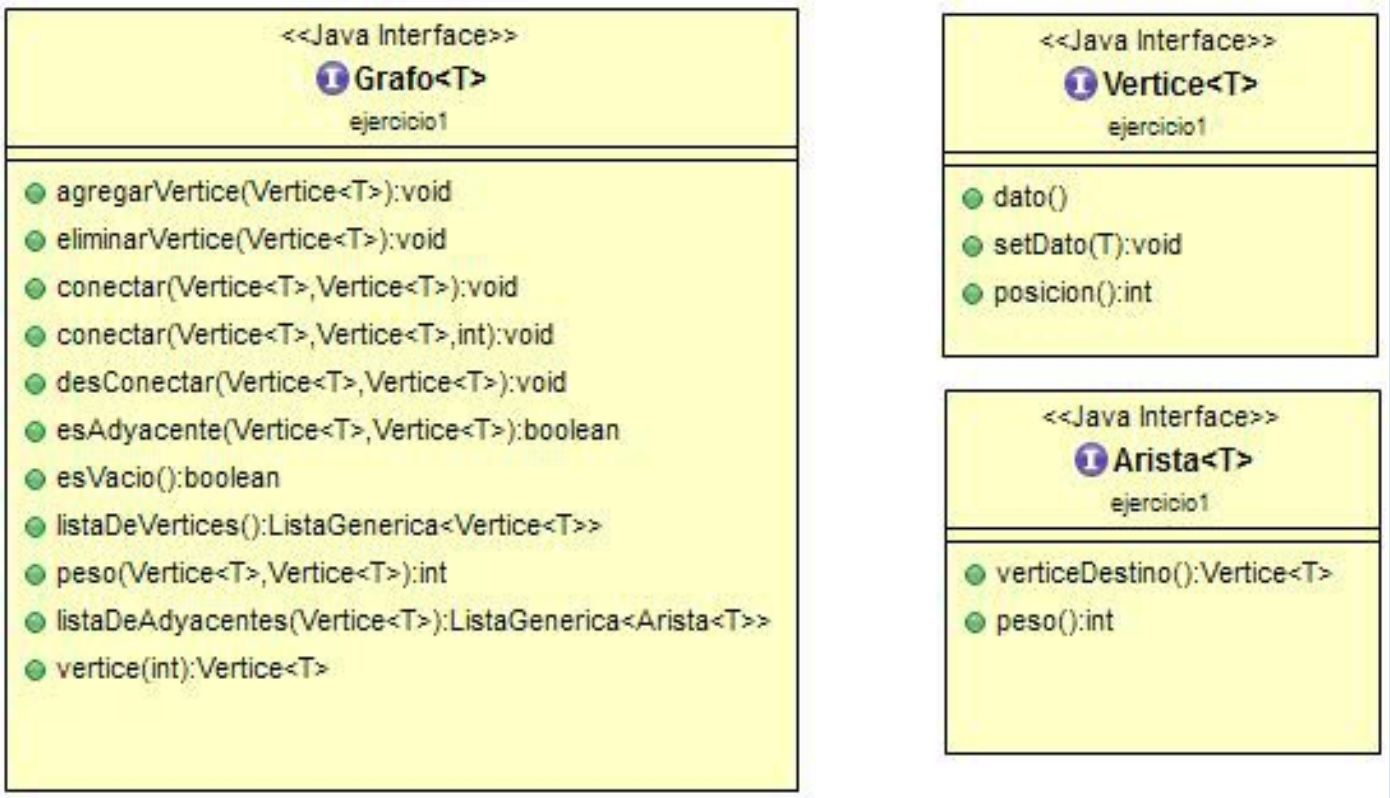
Por lo cuál en AyEd para implementar un Grafo definimos las siguientes tres interfaces:

- **Grafo**
- **Vertice**
- **Arista**

Que definen el comportamiento de un Grafo, independientemente si éste se representa usando una Lista de adyacencias o una Matriz de adyacencias.

Recordar:
Una interface java es una **colección de definiciones de métodos** sin implementación/cuerpo y de **declaraciones de variables de clase constantes**, agrupadas bajo un nombre. Una interface establece **qué** debe hacer la clase que la implementa, sin especificar el **cómo**.

Grafos - Interfaces



Grafos - Implementación

Las Representaciones típicas de grafos son dos:

- **Usando Matrices de Adyacencias y**
- **Usando Listas de Adyacencias**

Por lo cuál en AyED definimos las siguientes clases que permiten implementar las interfaces definidas usando estas dos representaciones:

- **GrafoImplListAdy**
- **VerticeImplListAdy**
- **GrafoImplMatrizAdy**
- **VerticeImplMatrizAdy**
- **AristaImpl**

Si ambas implementaciones soportan las mismas funcionalidades, bajo qué condiciones usaría una Matriz de Adyacencias en lugar de una Lista de Adyacencias para representar un grafo?????????

Grafos Implementación con Listas de Adyacencias

Grafos Implementación con Listas de Adyacencias


```
public class GrafoImplListAdy<T> implements Grafo<T> {

  private ListaGenerica<Vertice<T>> vertices = new ListaEnlazadaGenerica<Vertice<T>>();

  public void conectar (Vertice<T> origen, Vertice<T> destino) {
    ((VerticeImplListAdy<T>) origen).conectar(destino);
  }

  public void conectar (Vertice<T> origen, Vertice<T> destino, int peso) {
     ((VerticeImplListAdy<T>) origen).conectar(destino,peso);
  }

  public void desConectar (Vertice<T> origen, Vertice<T> destino) {
     ((VerticeImplListAdy<T>) origen).desconectar(destino);
  }

  public ListaGenerica<Arista<T>> listaDeAdyacentes (Vertice<T> v){
     return ((VerticeImplListAdy<T>) v).obtenerAdyacentes();
  }

  public ListaGenerica<Vertice<T>> listaDeVertices () {
     return vertices;
  }
```

Grafos Implementación con Listas de Adyacencias


```
public boolean esVacio() {
  return vertices.esVacia();
}

public int peso (Vertice<T> origen, Vertice<T> destino) {
  return ((VerticeImplListAdy<T>) origen).peso(destino);
}


public void agregarVertice (Vertice<T> v) {
   ….
}
public void eliminarVertice (Vertice<T> v) {
  …
}

}
```

Vertice implementado con Listas de Adyacencias


```
public class VerticeImplListAdy<T> implements Vertice<T> {
  private T dato;
  private int posicion;
  private ListaGenerica<Arista<T>> adyacentes;

  public VerticeImplListAdy (T d) {
     dato = d;
     adyacentes = new ListaEnlazadaGenerica<Arista<T>>();
  }

  public void conectar (Vertice<T> v){
     conectar(v, 1);
  }

  public void conectar(Vertice<T> v, int peso){
     Arista<T> a = new AristaImpl<T>(v, peso);
     if (!adyacentes.incluye(a)) adyacentes.agregarFinal(a);
  }
```

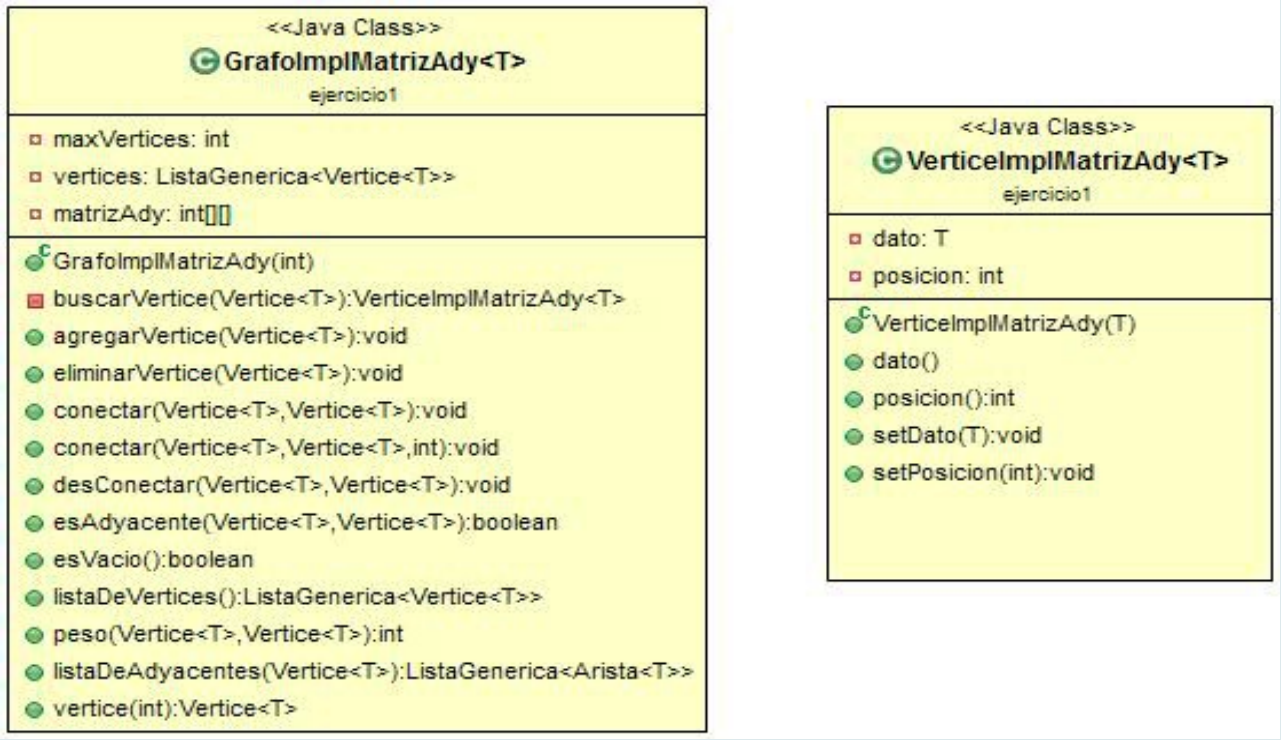
Vertice implementado con Listas de Adyacencias


```
…

public ListaGenerica<Arista<T>> obtenerAdyacentes(){
      return adyacentes;
}


public boolean esAdyacente (Vertice<T> v){
      this.obtenerAdyacentes().comenzar();
      while (!this.adyacentes.fin()){
         if (this.adyacentes.proximo().verticeDestino().posicion() == v.posicion()){
             return true;
         }
      }
      return false;
}

…
}
```

Implementación de la Interfaz Arista


```
public class AristaImpl<T> implements Arista<T> {

  private Vertice<T> destino;
  private int peso;

   public AristaImpl (Vertice<T> dest, int p){
      destino = dest;
      peso = p;
   }

   public Vertice<T> getDestino() {
      return destino;
   }

   public int peso() {
      return peso;
   }

}
```

Grafos Implementación con Matriz de Adyacencias



Grafos Implementación con Matriz de Adyacencias


```
public class GrafoImplMatrizAdy<T> implements Grafo<T> {
   private int maxVertices;
   private ListaGenerica<Vertice<T>> vertices;
   private int[][] matrizAdy;

   public GrafoImplMatrizAdy(int maxVert){
      this.maxVertices = maxVert + 1;// la lista arranca en 1, la posicion 0 no la usamos
      this.vertices = new ListaEnlazadaGenerica<Vertice<T>>();
      this.matrizAdy = new int[maxVertices][maxVertices];
   }

    public void  agregarVertice(Vertice<T> v) {
      VerticeImplMatrizAdy<T> v2 = (VerticeImplMatrizAdy<T>) v;
       if (! vertices.incluye(v2)){
           v2.setPosicion(vertices.tamanio());
           vertices.agregarFinal(v2);
        }
    }
    public void conectar (Vertice<T> origen, Vertice<T> destino, int peso) {
         matrizAdy[origen.posicion()][destino.posicion()] = peso;
    }
```

Grafos Implementación con Matriz de Adyacencias


```
public boolean esAdyacente(Vertice<T> origen, Vertice<T> destino) {
      if (vertices.incluye(origen) && vertices.incluye(destino)){
          return (!(matrizAdy[origen.posicion()][destino.posicion()]==0));
      }
      return false;
}

public void desConectar(Vertice<T> origen, Vertice<T> destino) {
      if (vertices.incluye(origen) && vertices.incluye(destino)){
         matrizAdy[origen.posicion()][destino.posicion()] = 0;
      }
}


public boolean esVacio() {
          return vertices.esVacia();
}

… // demás operaciones
}
```

}

Vertice Implementación con Matriz de Adyacencias


```
public class VerticeImplMatrizAdy<T> implements Vertice<T> {

   private T dato;
   private int posicion;


   public VerticeImplMatrizAdy(T d) {
     dato = d;
   }

   public T dato() {
      return dato;
   }

   public int posicion() {
      return posicion;
   }

   public void setPosicion(int posicion) {
      this.posicion = posicion;
   }


}
```