Tema 14 – Grafos y su Implementación en Java Parte I

Germán Moltó
Escuela Técnica Superior de Ingeniería Informática
Universidad Politécnica de Valencia

Tema 14 – Grafos y su Implementación en Java

Índice general:

Objetivos

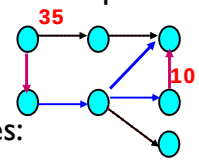
- Estudio de la Representación de una Relación Binaria entre los Datos de una Colección mediante la estructura Grafo y algunas de sus aplicaciones más significativas.
- Reutilizar las Estructuras de Datos empleadas en temas anteriores (Diccionario y Lista con Punto de Interés) y la implementación de las operaciones de Recorrido y cálculo de caminos mínimos sobre él (Modelos Cola y Cola de Prioridad).
- Implementación en Java de un Grafo, que supondrá el diseño de las clases Adyacente, Vertice, GrafoD, GrafoND y GrafoDEtiquetado (ubicadas en el paquete *grafos* de *estructurasDeDatos*).

Bibliografía

- Libro de M.A. Weiss, "Estructuras de Datos en Java" (Adisson-Wesley, 2000).
  - Capítulo 14, para conceptos sobre Grafos y Grafos Dirigidos
  - Capítulo 22, apartado 22.2.3 para el algoritmo de Dijkstra con Montículos de Emparejamiento
- Aho A.V., Hopcroft J.E., Ullman J.E. Estructuras de datos y Algoritmos. Addison-Wesley, 1988.
  - Capítulo 6 para conceptos sobre Grafos y Grafos Dirigidos

Motivación

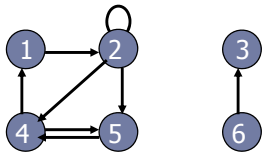
- En ocasiones, los elementos de una colección tienen una relación entre ellos que debe ser capturada mediante la Estructura de Datos empleada.



- Ejemplos posibles:
  - Elementos: Ciudades, Aeropuertos, Computadores de una Red, Puntos de un Plano, etc.
- Se pretende modelar:
  - Rutas entre ciudades, rutas aéreas, recorridos turísticos, tareas a realizar en un proyecto, etc.

Grafo Dirigido

- Un **Grafo Dirigido** (GD) es un Par G = (V,E)
  - V es un conjunto finito de Vértices (o Nodos o Puntos)
  - E es un conjunto de Aristas (o Arcos) dirigidas
- Arista: Par **ordenado** de Vértices (u,v)

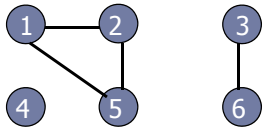


V = {1,2,3,4,5,6} |V| = 6

E = { (1,2), |E| = 8
(2,2), (2,4),(2,5),
(4,1),(4,5),
(5,4),
(6,3) }

Grafo No Dirigido

- Un **Grafo No Dirigido** (GND) es un Par G = (V,E)
  - V es un conjunto finito de Vértices
  - E es un conjunto de Aristas no Dirigidas
- Arista: Par **no ordenado** de Vértices (u,v) = (v,u)



V = {1,2,3,4,5,6} |V| = 6

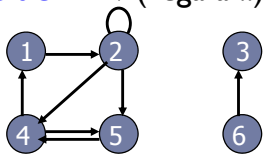
E = {(1,2), |E| = 4
(1,5),
(2,5),
(3,6) }

Grafo Etiquetado y Grafo Ponderado

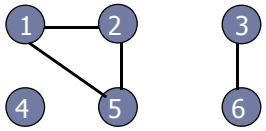
- Un **Grafo Etiquetado** es un grafo G = (V,E) sobre el que se define una función f: E → A, dónde A es un conjunto cuyas componentes se llaman Etiquetas.
- Un **Grafo Ponderado** es un Grafo Etiquetado (sus Aristas) con números Reales.
- También es posible definir la función de etiquetado para los Vértices, con lo que podemos asignar un nombre a cada Vértice.

Relaciones de Incidencia

- Sea G = (V,E) un **Grafo Dirigido**. Si (u,v) ∈ E, decimos que **Incide Desde** u (sale de ..) e **Incide En** v (llega a ..)
  - (2, 2) ∈ E, incide desde 2 e incide en 2
  - (1, 2) ∈ E, incide desde 1 e incide en 2
  - (2, 4) ∈ E, incide desde 2 e incide en 4
  - (2, 5) ∈ E, incide desde 2 e incide en 5

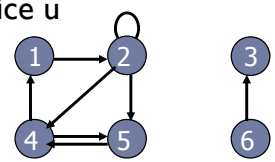


- Sea G = (V,E) un **Grafo no Dirigido**. Si (u,v) ∈ E, decimos que **Incide Sobre** u y v
  - (1,2) ∈ E, ó (2,1) incide sobre 1 y 2, ó 2 y 1
  - (2,5) ∈ E, ó (5,2) incide sobre 2 y 5, ó 5 y 2

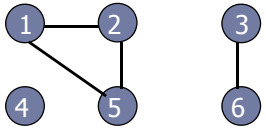


Relaciones de Adyacencia

- Sea G = (V,E) un Grafo. Si (u,v) ∈ E, decimos que el Vértice v **es Adyacente al** Vértice u
  - Ejemplo con el Vértice 2
    - 2 es Adyacente a 1
    - 1 no es Adyacente a 2



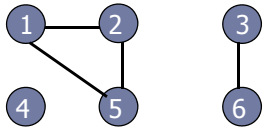
- En un Grafo no Dirigido la relación es simétrica:



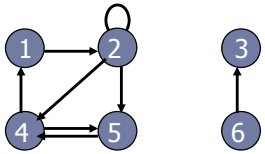
  - Ejemplo con el Vértice 3
    - 3 es Adyacente a 6
    - 6 es Adyacente a 3

Grado de un Vértice

- El **Grado de un Vértice** en un **Grafo no Dirigido** es el número de Aristas que Inciden sobre él (Vértices Adyacentes).
  - El Grado de 2 es 2



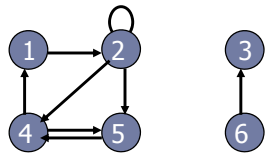
- El Grado de un Vértice en un Grafo Dirigido es la suma de:
  - El número de Aristas que salen de él (Grado de Salida)
  - El número de Aristas que entran en él (Grado de Entrada)
  - Ejemplos:
    - El Grado del Vertice 2 es 5
    - Grado de Entrada de 2 es 2
    - Grado de Salida de 2 es 3



  - El Grado de un Grafo es el de su Vértice de máximo Grado.

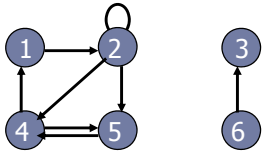
Caminos sobre Grafos

- Un **Camino** de longitud k desde u a u' en un grafo G=(V,E) es una secuencia de Vértices $<v_0,v_1,..,v_k>$ tal que:
  - $v_0 = u$ y $v_k = u'$
  - $\forall\ i : 1..k : (v_{i-1},v_i) \in E$
- La **Longitud** k del Camino …
  - **sin pesos** es el número de Aristas
  - **con pesos** es la suma de los Pesos de las Aristas del Camino
- Si hay un Camino P desde u hasta u', decimos que u' es **alcanzable** desde u vía P
  - Ejemplos:
    - 5 es alcanzable desde 1 vía <1,2,5>
    - 5 es alcanzable desde 1 vía <1,2,2,2,5>



Caminos Simples y Ciclos

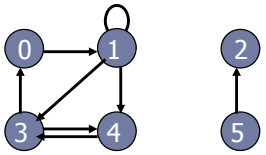
- Un Camino es **Simple** si todos sus Vértices intermedios son distintos. Si los extremos son iguales es un **Ciclo**.
  - Un **Bucle** es un ciclo de longitud 1
- En un **Grafo Dirigido** un Camino $<v_0,v_1,..,v_k>$ forma un Ciclo si:
  - $v_0 = v_k$
  - el Camino contiene al menos una Arista
- Un Grafo Dirigido es **Acíclico** si no contiene Ciclos (GDA)
  - Un Ciclo en este Grafo es <1, 2, 5, 4, 1>, de longitud 4.
  - El camino <2,2> es un bucle.



- Conceptos similares para los Grafos No Dirigidos.

Representación de un Grafo: Matriz de Adyacencia

- Un Grafo G=(V,E) se puede representar con una Matriz de |V|x|V| boolean. Si (u,v) ∈ E, G[u][v] = true; sino G[u][v] = false.
  - Memoria: $O(|V|^2)$
  - Tiempo de acceso: O(1)



	0	1	2	3	4	5
0	false	true	false	false	false	false
1	false	true	false	true	true	false
2	false	false	false	false	false	false
3	true	false	false	false	true	false
4	false	false	false	true	false	false
5	false	false	true	false	false	false

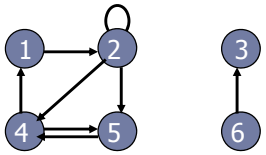
- Esta representación consume bastante memoria.
  - Útil para Grafos Densos (con muchas aristas).

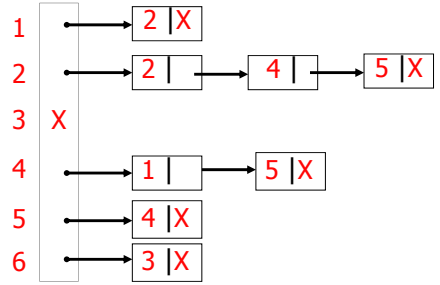
Cuestiones Rápidas: Representación con Matriz de Adyacencia

1. ¿Cómo saber si el grafo tiene bucles?
2. ¿Cómo saber el número de vértices?
3. ¿Cómo saber si hay algún vértice sin adyacentes?
4. ¿Cómo calcular el número de adyacentes de un vértice?
5. ¿Cómo sería la matriz de adyacencia de un grafo no dirigido?
6. ¿Cómo representar un Grafo con aristas ponderadas usando una matriz de adyacencias?

Representación de un Grafo: Listas de Adyacencia

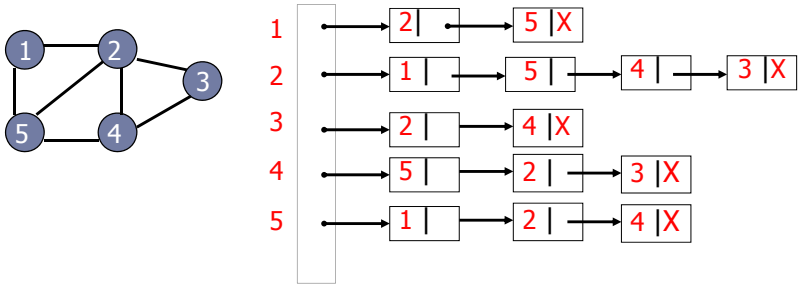
- Un **Grafo Dirigido** G=(V,E) se suele representar con un array de |V| Listas de Vértices: G[v] es una Lista de los Vértices Adyacentes a v (v ∈ V).
  - Memoria: O(|V|+|E|)
  - Tiempo de acceso: O(Grado de salida de G)





Representación de un Grafo: Listas de Adyacencia

- Un **Grafo No Dirigido** G=(V,E) se representa con un array de |V| Listas de Vértices: G[v] es una Lista de los Vértices Adyacentes a v (v ∈ V)
  - Memoria: O(|V|+2*|E|)
  - Tiempo de acceso: O(Grado de G)



Propuesta de Implementación de Grafos

- Grafo: Contiene los métodos comunes a cualquier implementación de un Grafo.
- GrafoD: Grafo dirigido con aristas ponderadas.
- GrafoND: Grafo no dirigido con aristas ponderadas
- GrafoDEtiquetado: Grafo dirigido con aristas ponderadas y vértices etiquetados.

La clase Java Grafo (I)

- Sobre la clase Grafo:
  - Su implementación irá creciendo conforme veamos métodos de recorrido, cálculo de caminos mínimos, etc.

```
package librerias.estructurasDeDatos.grafos;
public abstract class Grafo {
public Grafo() { ... }
public abstract int numVertices();
public abstract int numAristas();
public abstract boolean existeArista(int i, int j);
public abstract double pesoArista(int i, int j);
```


La clase Java Grafo (II)

```
public abstract void insertarArista(int i, int j);
public abstract void insertarArista(int i, int j, double p);
public abstract ListaConPI<Adyacente> adyacentesDe(int i);
public String toString(){
      String res = "" ;
      for (int  i = 1 ; i <= numVertices() ; i++) {
         res += "Vértice: " + i;
         ListaConPI<Adyacente> l = adyacentesDe(i);
         res += (l.esVacia()) ? " sin Adyacentes " :  " con Adyacentes: ";
         for (l.inicio(); !l.esFin() ; l.siguiente()) res +=  l.recuperar() + " ";
         res += "\n";
      } return res;
}
```

Implementa el método toString. Salida:
Vértice 1 con adyacentes 2 3
Vértice 2 sin adyacentes
...

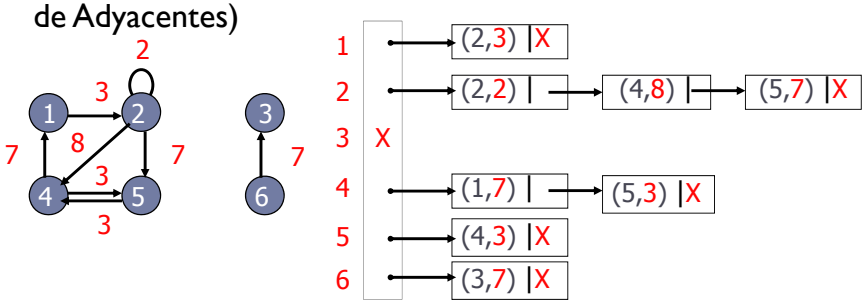
La clase Java Adyacente

```
package librerias.estructurasDeDatos.grafos;
class Adyacente{
    int destino;
    double peso;
    Adyacente(int codAdy, double p){ destino=codAdy; peso = p;}
    public String toString(){ return destino + "("+ peso + ")";}
}
```

- La clase Adyacente no se hace pública puesto que únicamente se utilizará dentro del paquete grafos (acceso friendly únicamente para el resto de clases del paquete).

Implementación de un Grafo Dirigido Ponderado: La Clase GrafoD

- *GrafoD* representa un Grafo
  - de Vértices **sin Etiquetar**
  - de Aristas **con Pesos**, Tripletes de int (origen, destino, coste)
- Mediante Listas de Adyacencia (array de |V| ListaConPI de Adyacentes)



La clase Java GrafoD (I)

```
package librerias.estructurasDeDatos.grafos;
import librerias.estructurasDeDatos.modelos.ListaConPI;
import librerias.estructurasDeDatos.lineales.LEGListaConPI;
public class GrafoD extends Grafo {
   protected int numV, numA;
   protected ListaConPI<Adyacente> elArray[];

@SuppressWarnings("unchecked")
   public GrafoD(int numVertices){
     numV = numVertices; numA=0;
     elArray = new ListaConPI[numVertices+1];
     for (int i=1; i<=numV; i++)elArray[i]= newLEGListaConPI<Adyacente>();
   }
```

Por convenio, los vértices se numeran desde 1..N y se almacenan en las posiciones 1..N del array.

La clase Java GrafoD (II)

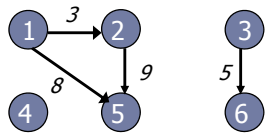
```
public int numVertices() { return numV;}
public int numAristas() { return numA; }
public boolean existeArista(int i, int j) {
     ListaConPI<Adyacente> l = elArray[i]; boolean esta=false;
     for (l.inicio(); !l.esFin()&& !esta; l.siguiente())
         if (l.recuperar().destino==j) esta =true;
     return esta;
   }
public double pesoArista(int i, int j) {
     ListaConPI<Adyacente> l = elArray[i];
     for (l.inicio(); !l.esFin(); l.siguiente())
         if (l.recuperar().destino==j) return l.recuperar().peso;
     return 0.0;
   }
```

La clase Java GrafoD (III)

```
public void insertarArista(int i, int j) {
   insertarArista(i,j,1);
}

public void insertarArista(int i, int j, double p) {
   if ( !existeArista(i,j) ) { elArray[i].insertar(new Adyacente(j,p)); numA++; }
}

public ListaConPI<Adyacente> adyacentesDe(int i) {return elArray[i];}

} /* Fin de la clase GrafoD */
```


Prueba de la clase GrafoD

```
public static void main(String args[]){
   GrafoD g = new GrafoD(6);
   g.insertarArista(1, 2, 3);
   g.insertarArista(1, 5, 8);
   g.insertarArista(2, 5, 9);
   g.insertarArista(3, 6, 5);
   System.out.println("El grafo es:  "+g.toString());
   System.out.println("existeArista(3,1) = " + g.existeArista(3,1));
   ListaConPI<Adyacente> l = g.adyacentesDe(1);
   System.out.println("Los adyacentes al vértice 1 son: ");
    for (l.inicio(); !l.esFin(); l.siguiente()) {
      System.out.println("(1, " + l.recuperar().destino + ")");
    }
}
```



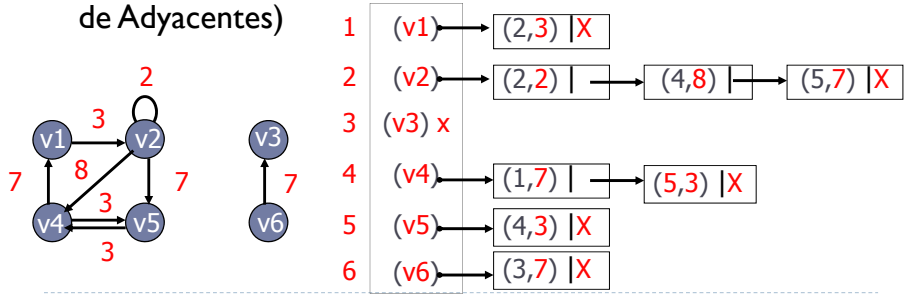
La clase Java GrafoND

```
package librerias.estructurasDeDatos.grafos;
import librerias.estructurasDeDatos.modelos.ListaConPI;
public class GrafoND extends GrafoD {
  public GrafoND(int numVertices) { super(numVertices); }

  public void insertarArista(int i, int j, int p) {
      if ( !existeArista(i,j) && i != j) {
        elArray[i].insertar(new Adyacente(j,p));
        elArray[j].insertar(new Adyacente(i,p));
       }
      numA++;
  }
} /* Fin de la clase GrafoND */
```

¿Cómo se implementaría el método insertarArista? No se permiten bucles.

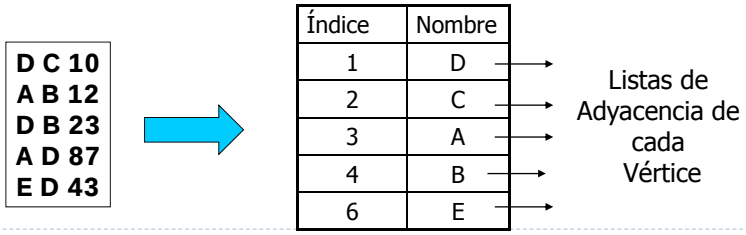
Implementación de Grafo Dirigido Ponderado con Vértices Etiquetados: Clase GrafoDEtiquetado

- La clase GrafoDEtiquetado representa un Grafo
  - de Vértices **con Etiquetas** (dato de tipo objeto), i.e. Pares (etiqueta/dato, n° vértice)
  - de Aristas **con Pesos**
- Mediante Listas de Adyacencia (array de |V| ListaConPI de Adyacentes)



Correspondencia entre Etiquetas de Vértice y su Código

- El constructor del Grafo especifica los vértices de los que va a constar el grafo.
  - Los vértices ya existen al crear el Grafo (aunque sin etiquetar).
- Para etiquetar un vértice, se utiliza el método:
  - public void etiquetarVertice(int codigo, E etiqueta)
- Para obtener el código de un vértice a partir de su etiqueta, utilizamos un **Diccionario<E,Integer>**.



La clase Java GrafoDEtiquetado (I)

```
package librerias.estructurasDeDatos.grafos;
import librerias.estructurasDeDatos.modelos.*;
import librerias.estructurasDeDatos.hash.TablaHashDiccionario;
import librerias.excepciones.ElementoNoEncontrado;
import java.util.ArrayList;
public class GrafoDEtiquetado<E> extends GrafoD{
   E etiquetas[];
   Diccionario<E, Integer> dicVertices;
   @SuppressWarnings("unchecked")
   public GrafoDEtiquetado(int numVertices) {
      super(numVertices);
      etiquetas = (E[]) new Object[numVertices+1];
      dicVertices = new TablaHashDiccionario<E, Integer>(numVertices);
   }
```


La clase Java GrafoDEtiquetado (II)

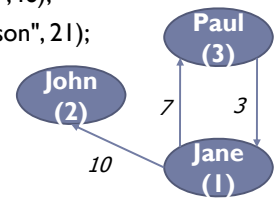
```
public boolean existeArista(E i, E j) {
      return existeArista(obtenerCodigo(i), obtenerCodigo(j));
   }
public double pesoArista(E i, E j) {
      return pesoArista(obtenerCodigo(i), obtenerCodigo(j));
   }
public void insertarArista(E i, E j) {
      insertarArista(obtenerCodigo(i), obtenerCodigo(j));
   }
 public void insertarArista(E i, E j, double p) {
     insertarArista(obtenerCodigo(i), obtenerCodigo(j), p);
 }
public ListaConPI<Adyacente> adyacentesDe(E i) {
      return adyacentesDe(obtenerCodigo(i));
   }
```


La clase Java GrafoDEtiquetado (III)

```
public void etiquetarVertice(int codigo, E etiqueta) {
      etiquetas[codigo] = etiqueta;
      dicVertices.insertar(etiqueta, codigo);
   }
public E obtenerEtiqueta(int codigo) {
      return etiquetas[codigo];
   }
public int obtenerCodigo(E etiqueta) {
      int codigo;
      try {
         codigo = dicVertices.recuperar(etiqueta).intValue();
      } catch (ElementoNoEncontrado e) { codigo = -1; }
      return codigo;
   }
```

Prueba de la clase GrafoDEtiquetado

```
public static void main(String args[]){
    GrafoDEtiquetado<Persona> g = new GrafoDEtiquetado<Persona>(3);
    Persona jane = new Persona("1123F","Jane Doe",25);
    Persona john = new Persona("5656M","John Doe",46);
    Persona paul = new Persona("8372G", "Paul Carlson", 21);
    g.etiquetarVertice(1,jane);
    g.etiquetarVertice(2,john);
    g.etiquetarVertice(3,paul);
    g.insertarArista(jane, john, 10);
    g.insertarArista(jane, paul, 7);
    g.insertarArista(paul, jane, 3);
    System.out.println("Grafo: " + g);
}
```



Grafo de afinidad ([0..10]) que siente una persona hacia las de su red social.