Ejercicios Tema 14

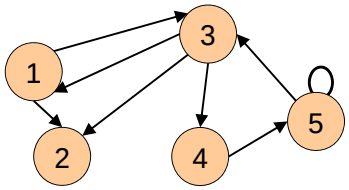
Ejercicios Adaptados de Apuntes y Exámenes de EDA
Germán Moltó
gmolto@dsic.upv.es
Estructuras de Datos y Algoritmos
Escuela Técnica Superior de Ingeniería Informática
Universidad Politécnica de Valencia

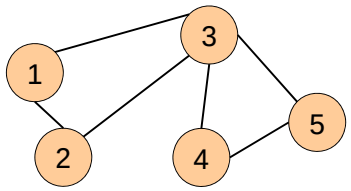
Propiedades

- Ejemplo: sea G = (V,E) un Grafo Dirigido con Pesos
  V={v0,v1, v2, v3, v4, v5, v6 },
  E={ (v0,v1, 2), (v0,v3, 1), (v1,v3, 3), (v1,v4, 10),
  (v3,v4, 2), (v3,v6, 4), (v3,v5, 8), (v3,v2, 2),
  (v2,v0, 4), (v2,v5, 5), (v4,v6, 6), (v6,v5, 1) }
- Se pide:
  1.- |V| y |E|
  2.- Vértices adyacentes a cada vi
  3.- Grado de cada vi y del Grafo
  4.- Caminos desde v0 al resto de Vértices, su longitud con y sin Pesos
  5.- Vértices alcanzables desde v0
  6.- Caminos mínimos desde v0 al resto de Vértices
  7.- ¿Tiene ciclos?

Representación de Grafo

- Representa los siguientes Grafos mediante una matriz de adyacencia y mediante listas de adyacencia.





Grado de Grafo (Matriz de Adyacencia)

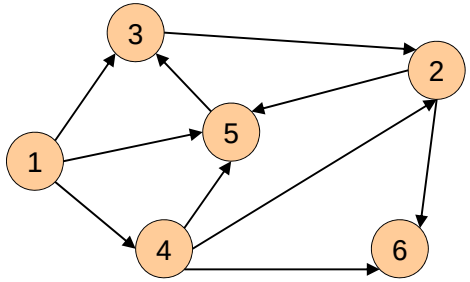
- Dado un grafo dirigido g y representado según una Matriz de Adyacencia:

```
package grafos;
public class GrafoMatriz{
  protected boolean matrizAdy[][];
  public GrafoMatriz (int numVertices){...}
  public void insertarArista (int origen, int destino, double coste){...};
  public String toString(){...}
}
```

• Se pide: Añadir a la clase GrafoMatriz un método gradoGrafo que calcule el grado del grafo.

Recorrido en Profundidad

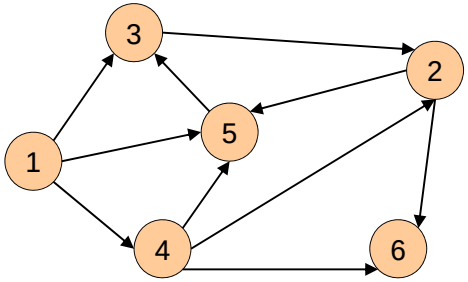
- Obtener la secuencia de vértices que se obtendría al hacer un recorrido en profundidad, partiendo desde el vértice 1 del siguiente grafo:



- Asumir que las aristas se recorren con el orden habitual de los enteros.

Recorrido en Anchura

- Obtener la secuencia de vértices que se obtendría al hacer un recorrido en anchura, partiendo desde el vértice 1 del siguiente grafo:



- Asumir que las aristas se recorren con el orden habitual de los enteros.

Recorridos en Grafo (1/2)

- Se ha inicializado un Grafo de 7 vértices a partir de la siguiente secuencia de Aristas:

e f 3
e g 2
f b 5
g f 1
g d 6
b d 5
b c 4
c a 7
a b 4
d a 5
d c 1

- Se pide:
  1. Dibujar el Grafo y su representación mediante Listas de Adyacencia.
- Asumir que:
  1. La inserción de una nueva arista se hace al principio de la correspondiente lista de adyacencia.
  2. Los vértices han sido previamente etiquetados en el orden en el que se han visto.

Recorridos en Grafo (2/2)

2. Escribir los nombres de los vértices del Grafo si se recorren en profundidad y que resultan de decodificar el array visitados que se obtiene al completar la traza del recorrido de la tabla.

Vértice	Visitados
	1 2 3 4 5 6 7 0 0 0 0 0 0 0

2. Escribir los nombres de los vértices del Grafo si se recorren en Amplitud y que resultan de decodificar el array visitados que se obtiene al completar la traza del recorrido de la tabla adjunta.

Vértice	Visitados	Cola
	1 2 3 4 5 6 7 0 0 0 0 0 0 0	

Nota: Los recorridos comenzarán desde el vértice situado en la posición 0 del vector.

Vértice Receptivo (Grafo No Dirigido)

- Dada la clase GrafoND que representa un Grafo no dirigido ponderado mediante Listas de Adyacencia, la clase Adyacente y la implementación de una Lista con Punto de Interés (ListaConPI) tal y como se ha visto en clase de teoría:
- Se desea añadir a la clase GrafoND un nuevo método, **getVerticeReceptivo**, que dado un Grafo obtenga el primer vértice sobre el que incidan todos los del Grafo, él mismo incluido.
  - Si no existe ningún vértice con esta característica se lanzará la excepción ElementoNoEncontrado.
  - Cada arista del grafo únicamente será visitada una vez.

Vértice Receptivo (Grafo Dirigido)

- Dada la clase GrafoDEtiquetado que representa un Grafo no dirigido ponderado mediante Listas de Adyacencia, la clase Adyacente y la implementación de una Lista con Punto de Interés (ListaConPI) tal y como se ha visto en clase de teoría:
- Se desea añadir a la clase GrafoDEtiquetado un nuevo método, **getVerticeReceptivo**, que obtenga la etiqueta del primer vértice sobre el que incidan todos los del Grafo, él mismo incluido.
  - Si no existe ningún vértice con esta característica se lanzará la excepción ElementoNoEncontrado.
  - Cada arista del grafo únicamente será visitada una vez.

Métodos de Grafo

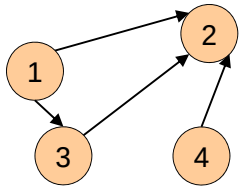
- Implementa los siguientes métodos de la clase Grafo:
  - Método para consultar el número total de aristas de un Grafo que inciden en un determinado vértice de código *codV*.
  - Método para consultar si un Grafo está vacío.
- Implementa los siguientes métodos de la clase GrafoD:
  - Método para consultar el grado de salida de un vértice, dado su código.
  - Método para consultar el grado de entrada de un vértice, dado su código.
  - Método para obtener el grado del grafo, utilizando los dos métodos anteriores.
  - Método para comprobar si un vértice dado es una fuente (vértice del que solo salen aristas).

Métodos de GrafoDEtiquetado

- Implementa los siguientes métodos de GrafoDEtiquetado:
  - Método para consultar si una arista dada (a partir de las etiquetas de los dos vértices) pertenece al grafo.

Vértice Sumidero de Todos los Demás

- Se dice que un Vértice de un Grafo Dirigido es un Sumidero si su grado de Entrada es mayor que cero y su grado de Salida es cero.
- **Se pide** diseñar en la clase GrafoD un método que, con coste mínimo, compruebe si un Vértice de código dado c es un Sumidero con grado de Entrada |V|-1, donde |V| es el número de Vértices del Grafo

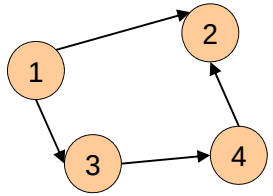


Grado Promedio de un Grafo

- Diseña en la clase GrafoD un método que, con coste mínimo, obtenga el grado promedio de un Grafo Dirigido o, equivalentemente, el promedio del grado de sus Vértices.

Grafo Dirigido Regular

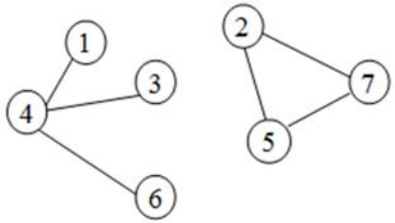
- Decimos que un **Grafo Dirigido** es **Regular** si todos sus Vértices tienen el mismo grado (con independencia de si es de salida o de entrada).
- **Se pide** añadir a la clase GrafoD un método que compruebe si un Grafo Dirigido es Regular con el menor coste Temporal posible (visitando cada arista del Grafo sólo una vez).
- NOTA: Es posible diseñar métodos auxiliares.



Componentes Conexas (1/2)

- Dado un Grafo **No Dirigido** se quiere diseñar un método **toStringCC** que obtenga en un **String** el número de Componentes Conexas que tiene junto con los Vértices que forman cada una de ellas. Por ejemplo, al aplicar tal método sobre el Grafo de la siguiente figura,

1: 4
2: 7,5
3: 4
4: 1,3,6
5: 2,7
6: 4
7: 2, 5



el **String** resultado sería: **"**Hay **2** Componentes Conexas y son:
[1 4 3 6]
[2 7 5]".

Componentes Conexas (2/2)

- Implementar la solución modificando los siguientes métodos.

```
// int ordenVisita y int[] visitados son Atributos de la clase
public int[]  toArrayDFS(){
  int[] res = new int[numVertices()+1];  visitados = new int[numVertices()+1];   ordenVisita = 1;
   for ( int  v=1; v<=numVertices(); v++ )
      if ( visitados[v] == 0 ) toArrayDFS(v, res);
   return res;
 }
protected void toArrayDFS(int v, int[] res){
   res[ordenVisita] = v; visitados[v] = ordenVisita++;
   ListaConPI<Adyacente> l = adyacentesDe(v);
   for ( l.inicio(); !l.esFin(); l.siguiente() ){
      int w = l.recuperar().destino;
      if ( visitados[w] == 0 ) toArrayDFS(w, res);
   }}
```