Ejresueltost14

Estructuras de datos y algoritmos (Universitat Politecnica de Valencia)

TEMA 14

La EDA Grafo y su Jerarquía Java

La jerarquía Grafo: modelo, implementación y aplicaciones sobre Grafos

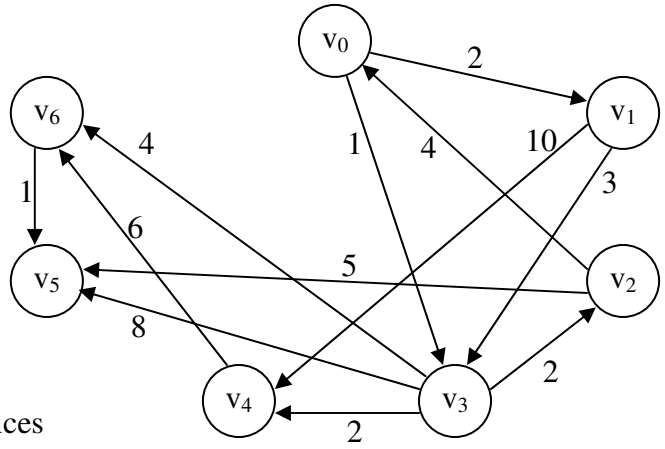
EJERCICIOS RESUELTOS

Ejercicio 1.- Sea $G = (V, E)$ un grafo dirigido con pesos:

$V = \{v_0, v_1, v_2, v_3, v_4, v_5, v_6\}$
$E = \{(v_0,v_1, 2), (v_0,v_3, 1), (v_1,v_3, 3), (v_1,v_4, 10), (v_3,v_4, 2), (v_3,v_6, 4), (v_3,v_5, 8), (v_3,v_2, 2), (v_2,v_0, 4), (v_2,v_5, 5), (v_4,v_6, 6), (v_6,v_5, 1)\}$

Se pide:

a) $|V|$ y $|E|$
b) Vértices adyacentes a cada uno de los vértices
c) Grado de cada vértice y del grafo
d) Caminos desde v_0 al resto de vértices, su longitud con y sin pesos
e) Vértices alcanzables desde v_0
f) Caminos mínimos desde v_0 al resto de vértices
g) ¿Tiene ciclos?



Solución:

a) $|V|$ y $|E|$

$|V| = 7$
$|E| = 12$

b) Vértices adyacentes a cada uno de los vértices

Adyacentes(v_0) = {v_1, v_3}
Adyacentes(v_1) = {v_3, v_4}
Adyacentes(v_2) = {v_0, v_5}
Adyacentes(v_3) = {v_2, v_4, v_5, v_6}
Adyacentes(v_4) = {v_6}
Adyacentes(v_5) = ∅
Adyacentes(v_6) = {v_5}

c) Grado de cada vértice y del grafo

Grado(v_0) = Grado(v_1) = Grado(v_2) = Grado(v_4) = Grado(v_5) = Grado(v_6) = 3
Grado(v_3) = Grado del grafo = 6

d) Caminos simples desde *v0* a *v6*, y su longitud con y sin pesos

Caminos (simples)	Longitud	Longitud con pesos
$\langle v_0, v_1, v_3, v_4, v_6\rangle$, $\langle v_0, v_1, v_3, v_6\rangle$, $\langle v_0, v_1, v_4, v_6\rangle$, $\langle v_0, v_3, v_4, v_6\rangle$, $\langle v_0, v_3, v_6\rangle$	4, 3, 3, 3, 2	13, 9 18, 9, 5

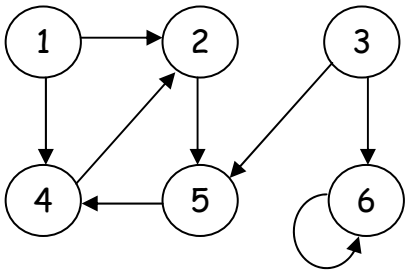
e) Vértices alcanzables desde *v0*
Todos

f) Caminos mínimos (con pesos) desde v0 al resto de vértices
$\langle v_0, v_1\rangle$, $\langle v_0, v_3, v_2\rangle$, $\langle v_0, v_3\rangle$, $\langle v_0, v_3, v_4\rangle$, $\langle v_0, v_3, v_6, v_5\rangle$, $\langle v_0, v_3, v_6\rangle$

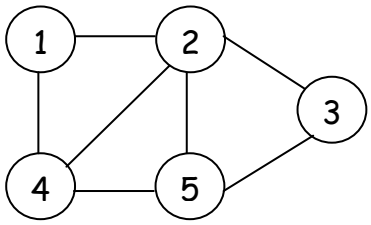
g) ¿Tiene ciclos?
Sí. Por ejemplo: $\langle v_0, v_3, v_2, v_0\rangle$

Ejercicio 2.- Representad los siguientes grafos mediante una matriz de adyacencia y mediante listas de adyacencia:

a)



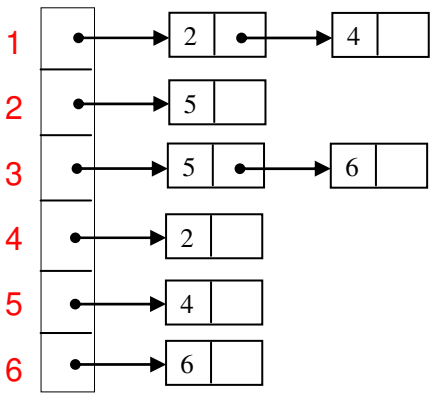
b)



Solución:

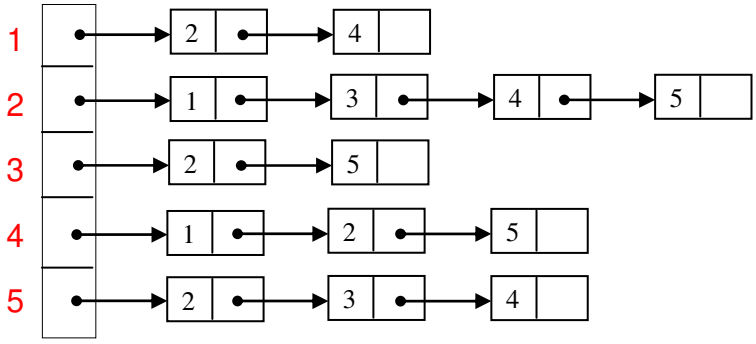
a)

	1	2	3	4	5	6
1	false	true	false	true	false	false
2	false	false	false	false	true	false
3	false	false	false	false	true	true
4	false	true	false	false	false	false
5	false	false	false	true	false	false
6	false	false	false	false	false	true

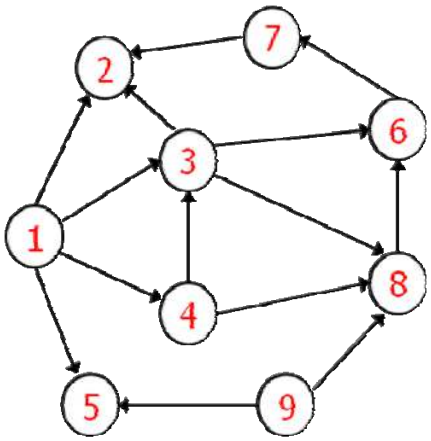


b)

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



Ejercicio 3.- Mostrar el resultado de imprimir por pantalla el recorrido en profundidad y en anchura del siguiente grafo:



Solución:

Nota: el orden de las aristas afecta al resultado de los recorridos. La solución que se muestra aquí asume que en la listas de adyacencia las aristas están ordenadas por el código del vértice destino de menor a mayor.

Profundidad (DFS): 1-2-3-6-7-8-4-5-9

Amplitud (BFS): 1-2-3-4-5-6-8-7-9

Ejercicio 4.- Definir los siguientes métodos en la clase *GrafoD*:

a) Consultar el grado de salida de un vértice dado
b) Consultar el grado de entrada de un vértice dado
c) Empleando los dos métodos anteriores, escribir un método que devuelva el grado del grafo
d) Comprobar si un vértice es *fuente*, es decir, si es un vértice del que sólo salen aristas
e) Comprobar si un vértice es un *sumidero* (i.e. un vértice al que sólo llegan aristas) al que llegan aristas de todos los demás vértices del grafo

Solución:

a) Consultar el grado de salida de un vértice dado

```
public int gradoDeSalida(int v) {
  int res = 0;
  ListaConPI<Adyacente> ady = elArray[v];
  for (ady.inicio(); !ady.esFin(); ady.siguiente()) res++;
  return res;
}
```

b) Consultar el grado de entrada de un vértice dado

```
public int gradoDeEntrada(int v) {
  int res = 0;
  for (int i = 1; i <= numV; i++) {
    ListaConPI<Adyacente> ady = elArray[i];
    for (ady.inicio(); !ady.esFin(); ady.siguiente())
      if (ady.recuperar().destino == v) {
        res++;
        ady.fin();
      }
  }
  return res;
}
```

c) Empleando los dos métodos anteriores, escribir un método que devuelva el grado del grafo

```
public int gradoGrafo() {
  int res = 0;
  for (int v = 1; v <= numV; v++) {
    int gradoV = gradoDeEntrada(v) + gradoDeSalida(v);
    if (gradoV > res) res = gradoV;
  }
  return res;
}
```

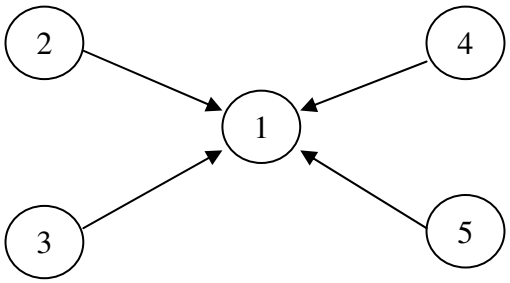
Nota: sin invocar a los dos métodos anteriores es posible implementar este método mucho más eficientemente.

d) Comprobar si un vértice es *fuente*, es decir, si es un vértice del que sólo salen aristas

```
public boolean esFuente(int v) {
  boolean res = true;
  for (int i = 1; i <= numV && res; i++) {
    ListaConPI<Adyacente> ady = elArray[i];
    for (ady.inicio(); !ady.esFin(); ady.siguiente())
      if (ady.recuperar().destino == v) {
        res = false;
        ady.fin();
      }
  }
  return res;
}
```

e) Comprobar si un vértice es un *sumidero* (i.e. un vértice al que sólo llegan aristas) al que llegan aristas de todos los demás vértices del grafo

Ejemplo: el vértice 1 es un sumidero de este tipo.




```
public boolean sumideroCompleto(int v) {
  boolean res = elArray[v].esVacia();
  for (int i = 1; i <= numV && res; i++) {
    if (i != v) {
      boolean llegaArista = false;
      ListaConPI<Adyacente> ady = elArray[i];
      for (ady.inicio(); !ady.esFin(); ady.siguiente())
        if (ady.recuperar().destino == v) {
          llegaArista = true;
          ady.fin();
        }
      if (!llegaArista) res = false;
    }
  }
  return res;
}
```