Tema + 7 + - + Ejercicios + Resueltos - Imp

Estructuras de datos y algoritmos (Universitat Politecnica de Valencia)

TEMA 7

Algoritmos sobre Grafos

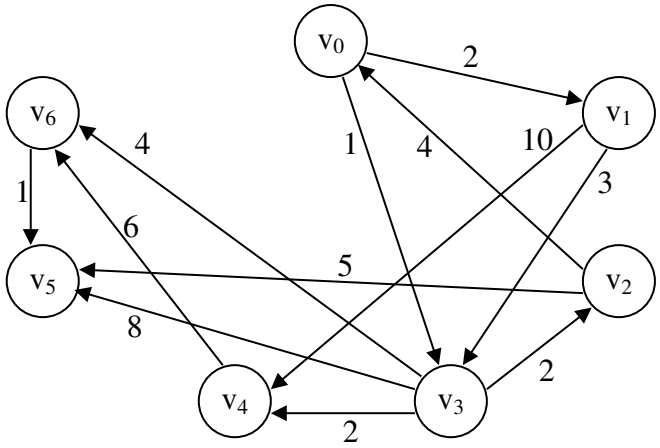
EJERCICIOS RESUELTOS

Ejercicio 1.- Sea $G = (V, A)$ 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) $|V| = 7$
$|A| = 12$

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) = $\varnothing$
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

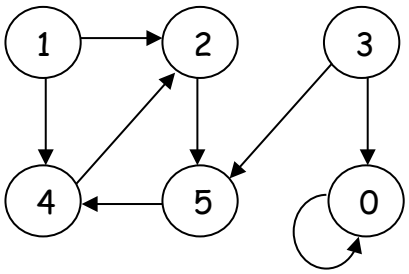
e) Vértices alcanzables desde *v0*
Todos

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

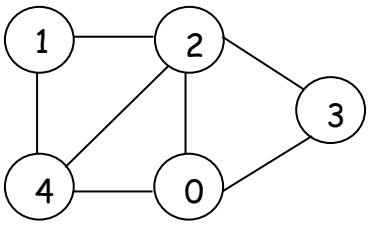
g) ¿Tiene ciclos?
Sí. Por ejemplo: $\langle v_0, v_3, v_2, v_0\rangle$

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

a)



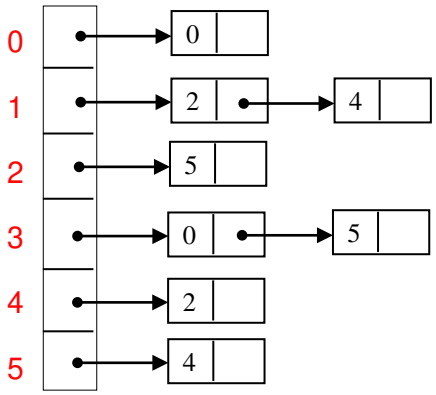
b)



Solución:

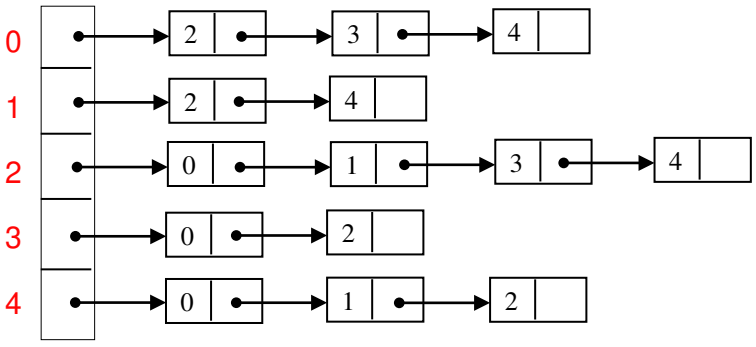
a)

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

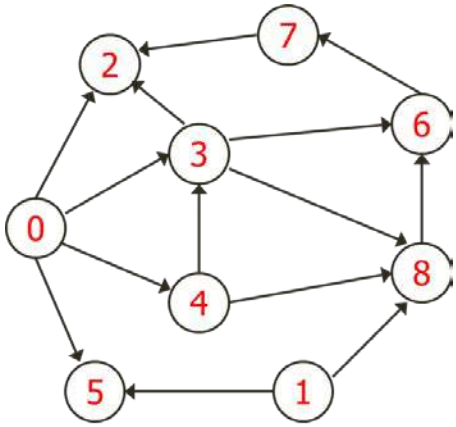


b)

	0	1	2	3	4
0	false	false	true	true	true
1	false	false	true	false	true
2	false	true	false	true	true
3	false	false	true	false	false
4	true	true	true	false	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): 0-2-3-6-7-8-4-5-1

Amplitud (BFS): 0-2-3-4-5-6-8-7-1

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

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) {
  return elArray[v].talla();
}
```

b) Consultar el grado de entrada de un vértice dado

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

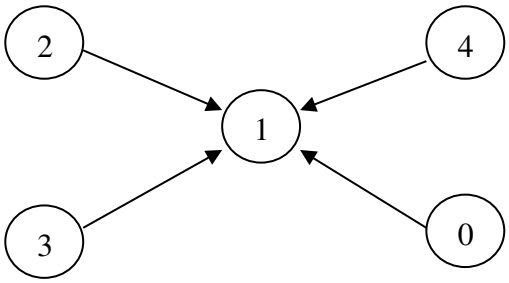
Nota: sin invocar a los dos métodos anteriores es posible implementar este método mucho más eficientemente.

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 = !elArray[v].esVacia();
  for (int i = 0; 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 = 0; 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;
}
```

Ejercicio 5.- Implementa un método en la clase *Grafo* que compruebe si un vértice es alcanzable desde otro vértice dado.

Solución:

```
public boolean esAlcanzable(int vOrigen, int vDestino) {
  visitados = new boolean[numVertices()];
  return esAlcanzableRec(vOrigen, vDestino);
}

private boolean esAlcanzableRec(int vActual, int vDestino) {
  if (vActual == vDestino) return true;
  visitados[vActual] = true;
  ListaConPI<Adyacente> ady = adyacentesDe(vActual);
  for (ady.inicio(); !ady.esFin(); ady.siguiente()) {
    int vSiguiente = ady.recuperar().destino;
    if (visitados[vSiguiente] == 0 && esAlcanzableRec(vSiguiente, vDestino))
       return true;
  }
  return false;
}
```

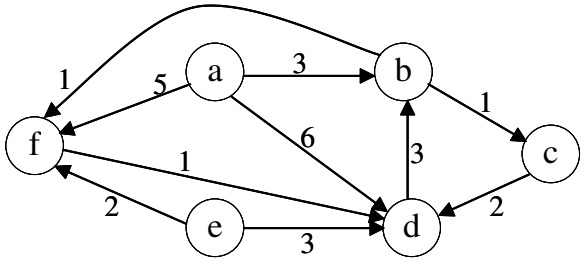
Ejercicio 6.- Un grafo transpuesto *T* de un grafo *G* tiene el mismo conjunto de vértices pero con las direcciones de las aristas en sentido contrario, es decir, que una arista (u, v) en *G* se corresponde con una arista (v, u) en *T*.
Diseña un método en la clase *GrafoDirigido* que permita obtener su grafo transpuesto:

```
public GrafoDirigido grafoTranspuesto();
```

Solución:

```
private GrafoDirigido grafoTranspuesto() {
  GrafoDirigido res = new GrafoDirigido(numV);
  for (int i = 0; i < numV; i++) {
    ListaConPI<Adyacente> ady = elArray[i];
    for (ady.inicio(); !ady.esFin(); ady.siguiente()) {
      Adyacente a = ady.recuperar();
      res.insertarArista(a.destino, i, a.peso);
    }
  }
  return res;
}
```

Ejercicio 7.- Los vértices del siguiente grafo representan personas (Ana, Begoña, Carmen, Daniel, Eliseo y Francisco) y las aristas indican si una persona tiene el número de móvil de otra. El peso de una arista es el coste de enviar un SMS (por ejemplo, Ana puede enviar un SMS a Begoña por 3 céntimos).
Haz una traza de Dijkstra para averiguar cuál sería la forma más barata de que Ana le haga llegar un SMS a Francisco.



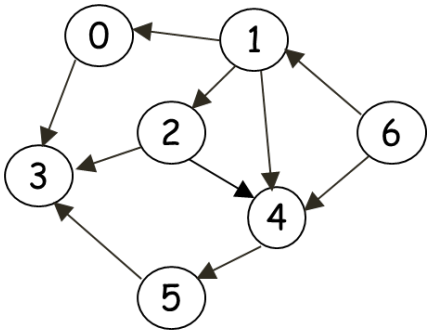
Solución:

Asumimos que los códigos de los vértices son: a = 0, b = 1, c = 2, d = 3, e = 4, f = 5.

V	qPrior	visitados	distanciaMin	caminoMin
	(0,0)	F F F F F F	0 ∞ ∞ ∞ ∞ ∞	-1 -1 -1 -1 -1 -1
0	**(1,3)**,(3,6),(5,5)	T F F F F F	0 3 ∞ 6 ∞ 5	-1 0 -1 0 -1 0
1	(3,6),(5,5),**(2,4)**,(5,4)	T T F F F F	0 3 4 6 ∞ 4	-1 0 1 0 -1 1
2	(3,6),(5,5),**(5,4)**	T T T F F F	0 3 4 6 ∞ 4	-1 0 1 0 -1 1
5	(3,6),(5,5),**(3,5)**	T T T F F T	0 3 4 5 ∞ 4	-1 0 1 5 -1 1
3	(3,6),**(5,5)**	T T T T F T	0 3 4 5 ∞ 4	-1 0 1 5 -1 1
5	**(3,6)**			
3	∅			

La forma más barata de que Ana le haga llegar un SMS a Francisco es: Ana → Begoña → Francisco.

Ejercicio 8.- Siguiendo el método *ordenacionTopologica*, mostrar la ordenación topológica resultante para el siguiente grafo dirigido acíclico:



¿La ordenación obtenida es única? En caso negativo mostrar otra ordenación válida.

Solución:

Ordenación obtenida:

$6 \rightarrow 1 \rightarrow 2 \rightarrow 4 \rightarrow 5 \rightarrow 0 \rightarrow 3$

No es la única ordenación posible. Ejemplo:

$6 \rightarrow 1 \rightarrow 0 \rightarrow 2 \rightarrow 4 \rightarrow 5 \rightarrow 3$