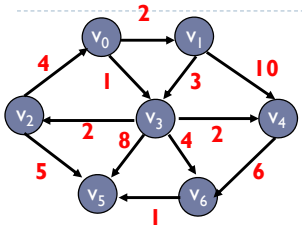
Soluciones Ejercicios Tema 14

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 (1/2)



|V| = 7 |E| = 12

grado de v_0: 2 + 1
grado de v_1: 2 + 1
grado de v_2: 2 + 1
grado de v_3: 4 + 2
grado de v_4: 1 + 2
grado de v_5: 0 + 3
grado de v_6: 1 + 2

Grado del grafo: 6

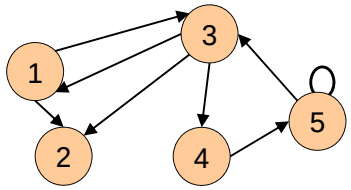
adyacentes a v_0: v_1, v_3
adyacentes a v_1: v_4, v_3
adyacentes a v_2: v_0, v_5
adyacentes a v_3: v_4, v_6, v_5, v_2
adyacentes a v_4: v_6
adyacentes a v_5: 0
adyacentes a v_6: v_5

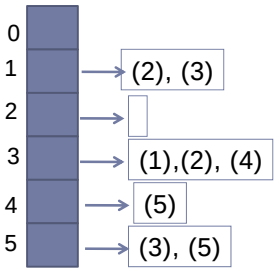
Propiedades (2/2)

Camino desde v_0 a v_0 : no hay → longitud 0
Camino desde v_0 a v_1 : $\langle v_0,v_1\rangle$, de longitud 1
Caminos desde v_0 a v_2 : $\langle v_0,v_1,v_3,v_2\rangle$ de longitud 3
$\langle v_0,v_3,v_2\rangle$ de longitud 2
Caminos desde v_0 a v_4 : $\langle v_0,v_1,v_4\rangle$ de longitud 2
$\langle v_0,v_3,v_4\rangle$ de longitud 2
$\langle v_0,v_1,v_3,v_4\rangle$ de longitud 3
$\langle v_0,v_3,v_2,v_0,\ldots.\rangle$ de longitud
...

- Si que tiene ciclos, ej.: $<v_2,v_0,v_3,v_2>$

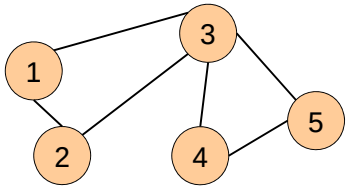
Representación de Grafo (I)

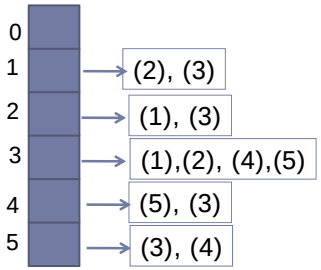




	1	2	3	4	5
1		T	T		
2					
3	T	T		T	
4					T
5			T		T

Representación de Grafo (II)





	1	2	3	4	5
1		T	T		
2	T		T		
3	T	T		T	T
4			T		T
5			T	T	

Grado de Grafo (Matriz de Adyacencia)

```
public int gradoGrafo() {
    int numVertices = matriz.length;
    int gradoMax=0;
    for ( int i=0; i<numVertices ; i++ ) {
        int grado = 0;
        for ( int j=0; j<numVertices ; j++ ) {
          if ( matriz[i][j] ) grado++;
          if ( matriz[j][i] ) grado++;
        }
         if ( grado > gradoMax ) gradoMax = grado;
     }
     return gradoMax;
  }
```


Recorrido en Profundidad

Vértice	Visitados
	1 2 3 4 5 6 0 0 0 0 0 0
1: 3 4 5	1 0 0 0 0 0
3: 2	1 0 2 0 0 0
2: 5 6	1 3 2 0 0 0
5: 3	1 3 2 0 4 0
6: {}	1 3 2 0 4 5
4: 2 5 6	1 3 2 6 4 5

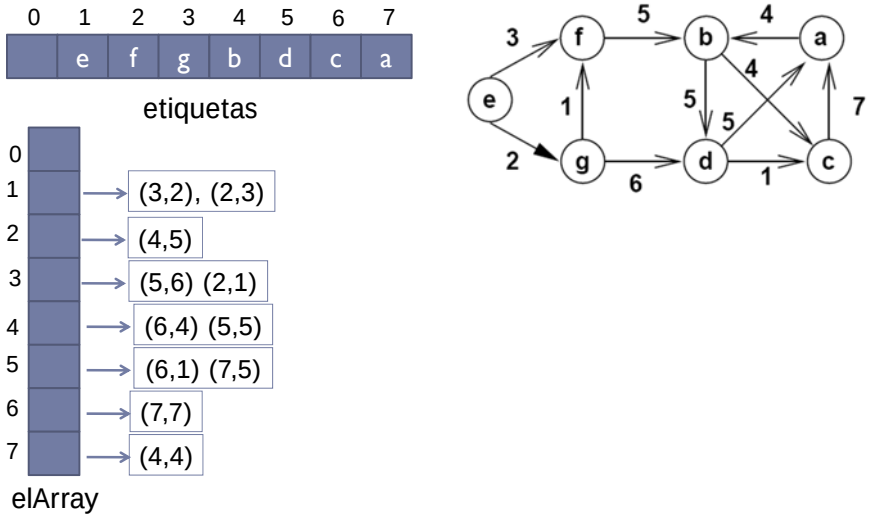
- La secuencia de vértices recorrida: 1 3 2 5 6 4

Recorrido en Anchura

Vértice	Visitados	Cola
	1 2 3 4 5 6 0 0 0 0 0 0	
1 : 3 4 5	1 0 2 3 4 0	3 4 5
3: 2	1 5 2 3 4 0	4 5 2
4: 2 5 6	1 5 2 3 4 6	5 2 6
5: 3	1 5 2 3 4 6	2 6
2: 5 6	1 5 2 3 4 6	6
6: {}	1 5 2 3 4 6	

- La secuencia de vértices recorrida: 1 3 4 5 2 6

Recorridos en Grafo (1/3)



Recorridos en Grafo (2/3)

- Recorrido en Profundidad (DFS)

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

- La secuencia de vértices (DFS) es: e g d c a b f

Recorridos en Grafo (3/3)

- Recorrido en Anchura (BFS)

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

- La secuencia de vértices (BFS) es: e g f d b c a

Vértice Receptivo (Grafo No Dirigido) (1/2)

- Asumiendo que trabajamos con un Grafo No Dirigido, la estrategia es obtener el grado de cada vértice y en cuanto encontremos uno cuyo grado es igual al número de vértices, detenemos la búsqueda.
- Si la búsqueda alcanza el final del vector de vértices sin encontrar un vértice que cumpla la propiedad, lanzaremos la excepción ElementoNoEncontrado.

Vértice Receptivo (Grafo No Dirigido) (2/2)

```
public int getVerticeReceptivo() throws ElementoNoEncontrado{
  int i = 1, naris;  boolean esta = false;
  int nVertices = numVertices();
  while (i <= nVertices && !esta ) {
    naris = 0;  ListaConPI<Adyacente> lista = elArray[i];
    for (lista.inicio(); !lista.esFin(); lista.siguiente()) naris++;
    if (naris == nVertices) esta = true;
    else i++;
  }
  if (!esta) throw new ElementoNoEncontrado("No esta");
  else return i;  }
```


Vértice Receptivo (Grafo Dirigido) (1/3)

- Si consideramos el Grafo como Dirigido, entonces:
- Solución estructurada en dos pasos:
  - Paso 1: Calcular el **grado de entrada** de todos los vértices del Grafo, recorriendo cada Lista de Adyacencia (uso de un vector auxiliar gradoEntradaVertice).
  - Paso 2: Búsqueda del primer elemento de gradoEntradaVertice cuyo valor sea numVertices.

Vértice Receptivo (Grafo Dirigido) (2/3)

```
public E getVerticeReceptivo() throws ElementoNoEncontrado{
   int gradoEntradaVertice[] = new int[numVertices() + 1];
   for ( int i = 1; i <= numVertices(); i++) gradoEntradaVertice[i] = 0;

   //Paso 1: Cálculo del grado de entrada de cada vértice
   for (int i = 1; i <= numVertices(); i++) {
         ListaConPI<Adyacente> l = elArray[i];
         for (l.inicio(); !l.esFin(); l.siguiente()) {
           gradoEntradaVertice[l.recuperar().destino]++;
     }
   }
...
```


Vértice Receptivo (Grafo Dirigido) (3/3)

```
   //Paso 2: busqueda del Vertice con grado de entrada numVertices
   for (int i = 1; i < numVertices(); i++){
      if ( gradoEntradaVertice[i] == numVertices()) return
   etiquetas[i];
    }
   throw new ElementoNoEncontrado("No hay ningún Vértice
   sobre el que inciden todos");

}
```

Métodos de la Clase Grafo

```
public int numeroDeAristas(int codV) {
  int naris = 0;
  for (int i = 1; i <= numVertices(); i++) {
        ListaConPI<Adyacente> ady = adyacentesDe(i);
        for (ady.inicio(); !ady.esFin(); ady.siguiente())
          if (ady.recuperar().destino == codV) naris++;
   }
   return naris;
 }
public boolean esVacio() { return numVertices()== 0);}
```


Métodos de la clase GrafoD (1/2)

```
public int gradoSalida(int codV) {
   int grado = 0;
   ListaConPI<Adyacente> ady = adyacentesDe(codV);
   for (ady.inicio(); !ady.esFin(); ady.siguiente()) grado++;
   return grado;
}
public int gradoEntrada(int codV){
     int grado = 0;
     for (int i = 1; i <= numVertices(); i++){
        ListaConPI<Adyacente> l = this.elArray[i];
        for (l.inicio(); !l.esFin();  l.siguiente())
           if (l.recuperar().destino == codV) {grado++;  break;}
      } return grado;
   }
```

Posible optimización: Como desde un vértice no puede llegar más de una arista a otro vértice, en cuanto encontramos dicha arista no es necesario seguir recorriendo la lista de adyacencias.

Métodos de la clase GrafoD (2/2)

```
public int grado(){
      int maxGrado = 0, grado;
      for (int i = 1; i <= numVertices(); i++){
        grado = gradoEntrada(i) + gradoSalida(i);
        if (grado > maxGrado) maxGrado = grado;
      } return maxGrado;  }

public boolean esFuente(int codV){
    for (int i = 1; i <= numVertices(); i++){
       ListaConPI<Adyacente> l = this.elArray[i];
       for (l.inicio(); !l.esFin(); l.siguiente()) {
          if (l.recuperar().destino == codV) return false;
        }
      } return true; }
```


Vértice Sumidero de Todos los Demás

```
public boolean esSumideroDeTodos(int c){
  if ( !this.elArray[c].esVacia() ) return false;
  boolean estaC = true;
  for ( int i = 1 ; i <= numVertices() && estaC; i++){
    ListaConPI<Adyacente>  l = this.elArray[i];
    for (l.inicio(); !l.esFin() && !(l.recuperar().destino == c); l.siguiente()){;}
    if (  l.esFin() &&  i  != c  )  estaC = false;
  }
  return estaC;
}
```

- La lista de adyacencia de *c* deberá estar vacía y *c* debe estar en la lista de adyacencia de todos los demás vértices.

Grado Promedio de un Grafo

- Para tener una implementación de coste espacial y temporal mínimo, hay que fijarse que, en un Grafo dirigido se cumple la siguiente propiedad:
  - La suma del grado de salida de todos los vértices, equivale a la suma del grado de entrada de todos los vértices.
- Por lo tanto, gradoEntrada+ gradoSalida = 2*numAristas

```
public double gradoMedio(){
    return ( 2.0 * numAristas() / numVertices());
 }
```


Grafo Dirigido Regular (1/2)

```
public boolean esRegular(){
        // Se calcula el grado de los Vértices como paso previo
        int[] gradoV = gradosVertices();
        // A continuación, se comprueba si es Regular
        boolean esRegular = true; int gradoV0 = gradoV[1];
        for ( int i = 2; i < elArray.length && esRegular; i++ )
                esRegular = ( gradoV[i] == gradoV0 );
        return esRegular;
    }
```


Grafo Dirigido Regular (2/2)

```
protected int[] gradosVertices(){
    int [] gradoV = new int[numVertices() + 1];
        for ( int i = 1; i < elArray.length; i++ ){
            ListaConPI<Adyacente> aux = elArray[i];
            for ( aux.inicio(); !aux.esFin(); aux.siguiente() ){
              // actualización del grado de Salida del Vértice de código i
              gradoV[i]++;
              // actualización del grado de Entrada del Vértice
              gradoV[aux.recuperar().destino]++;
            }
        }
        return gradoV;
    }
```


Componentes Conexas

```
public String toStringCC(){
 visitados = new int[numVertices()+1];
 String res=""; int nCC=0;
 for ( int  v=1; v<=numVertices(); v++)
   if (  visitados[v] == 0  ) {
                nCC++; res += "["+ toStringCC(v)+"]\n";
    }  return "Hay "+nCC+" Componentes Conexas y son:\n"+res;
}
protected String toStringCC(int v){
    String res=""+v; visitados[v] = 1;
     ListaConPI<Adyacente> l = adyacentesDe(v);
     for ( l.inicio(); !l.esFin(); l.siguiente() ){
        int w = l.recuperar().destino;
       if ( visitados[w]  ==  0 ) res += " "+recorridoDFS (w);
      } return res; }
```