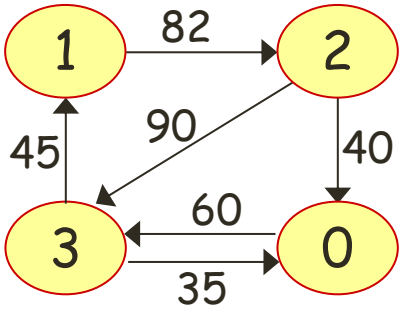
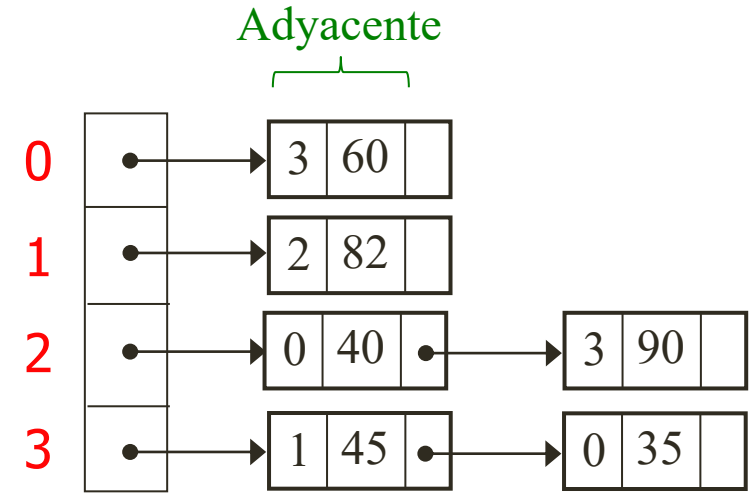
3. Implementación

La clase Adyacente

```
class Adyacente {
  int destino;        // Vértice destino de la arista
  double peso;        // Peso de la arista
  public Adyacente(int d, double p){ destino = d; peso = p; }
  public String toString(){ return destino + "(" + peso + ")"; }
}
```

Ejemplo:





3. Implementación

La clase GrafoDirigido (1/4)

```
// Implementación de un Grafo Dirigido
public class GrafoDirigido extends Grafo {

  // Número de vértices y aristas
  protected int numV, numA;
   // El array de listas con los adyacentes de cada vértice
  protected ListaConPI<Adyacente> elArray[];

  // Construye un Grafo con un numero de vertices dado
  @SuppressWarnings("unchecked")
  public GrafoDirigido(int numVertices) {
    numV = numVertices;
    numA = 0;
    elArray = new ListaConPI[numVertices];
    for (int i = 0; i < numV; i++)
      elArray[i] = new LEGListaConPI<Adyacente>();
  }
```

3. Implementación

La clase GrafoDirigido (2/4)

```
// Consultores
public int numVertices() { return numV; }
public int numAristas()  { return numA; }

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

3. Implementación

La clase GrafoDirigido (3/4)

```
// Consultores
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;
}
```

3. Implementación

La clase GrafoDirigido (4/4)

```
// Inserción de aristas
public void insertarArista(int i, int j) {
  insertarArista(i, j, 1.0);  // El peso por defecto es 1.0
}
public void insertarArista(int i, int j, double p) {
  if (!existeArista(i,j)) {
    elArray[i].insertar(new Adyacente(j,p));
    numA++;
  }
}
```

3. Implementación

Ejercicios

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

a) Consultar el grado de salida de un vértice dado

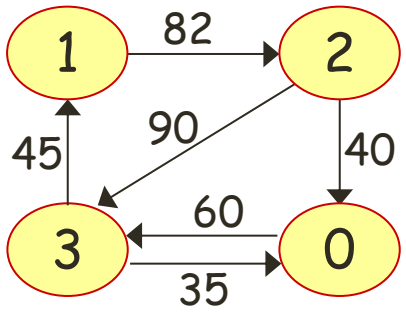
b) Consultar el grado de entrada de un vértice dado

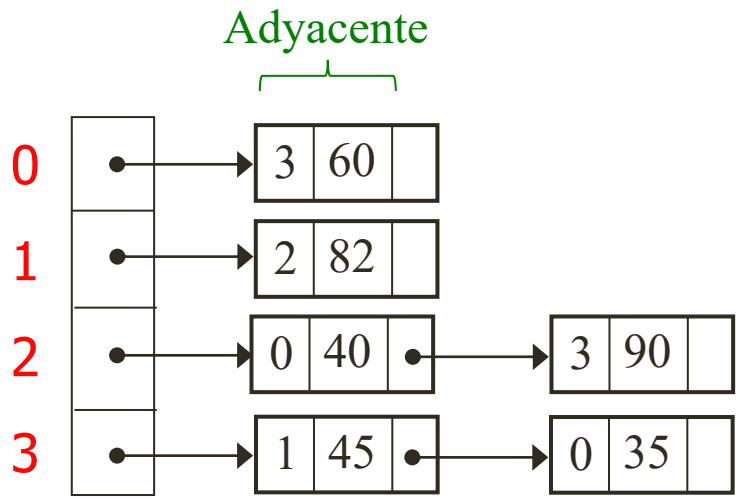
c) Empleando los dos métodos anteriores, escribir un método que devuelva el grado del grafo

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

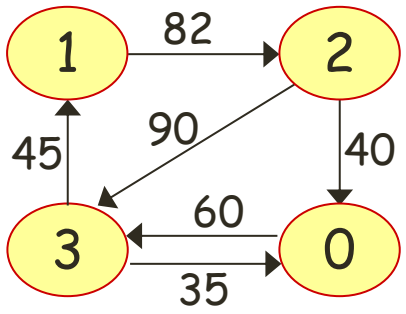
a) Consultar el grado de salida de un vértice dado

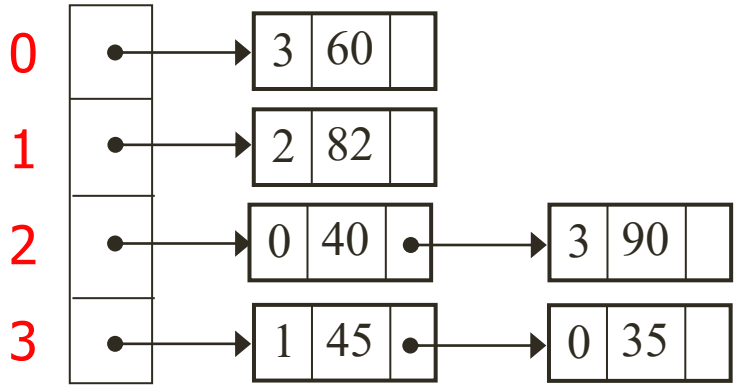





```
public int gradoDeSalida(int v) {
  return elArray[v].talla();
}
```

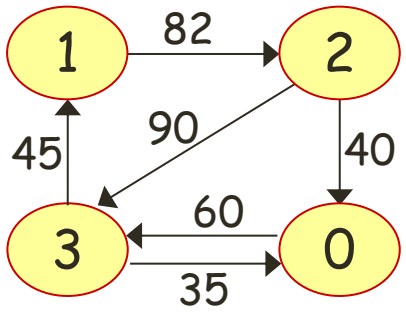
b) Consultar el grado de entrada de un vértice dado

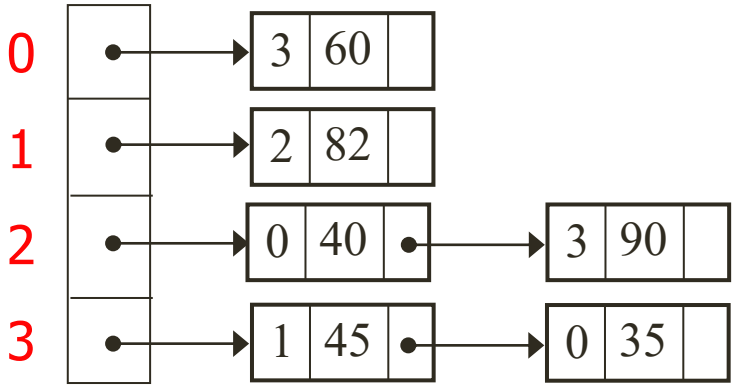





```
public int gradoDeEntrada(int v) {
  int res = 0;
  for (int i = 0; i < numV; i++) {
    ListaConPI<Adyacente> ady = elArray[i];
    for (ady.inicio(); !ady.esFin(); ady.siguiente())
      if (ady.recuperar().destino == v) {
        res++;
        ady.fin();
      }
  }
  return res;
}
```

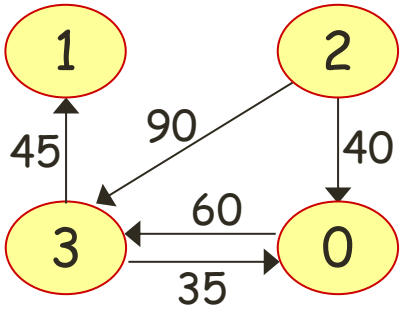
c) Empleando los dos métodos anteriores, escribir un método que devuelva el grado del grafo

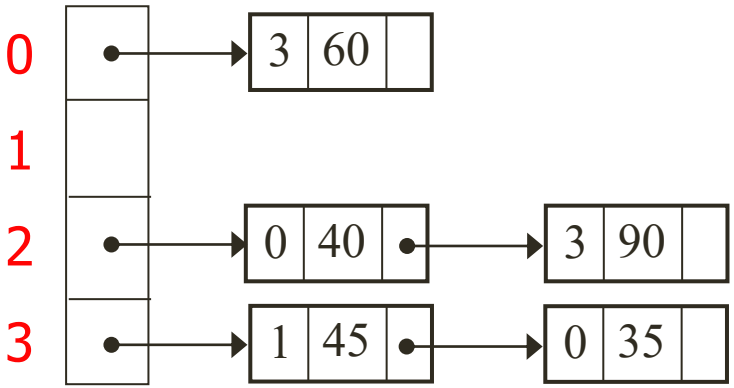





```
public int gradoGrafo() {
  int res = 0;
  for (int v = 0; v < numV; v++) {
    int gradoV = gradoDeEntrada(v) + gradoDeSalida(v);
    if (gradoV > res) res = gradoV;
  }
  return res;
}
```

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

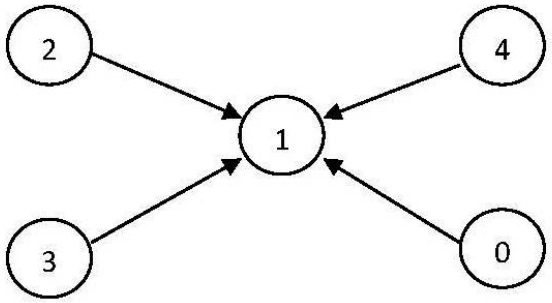





```
public boolean esFuente(int v) {
  boolean res = !elArray[v].esVacia();
  for (int i = 0; i < numV && res; i++) {
    ListaConPI<Adyacente> ady = elArray[i];
    for (ady.inicio(); !ady.esFin(); ady.siguiente())
      if (ady.recuperar().destino == v) {
        res = false;
        ady.fin();
  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

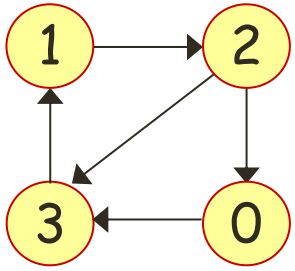
3. Implementación

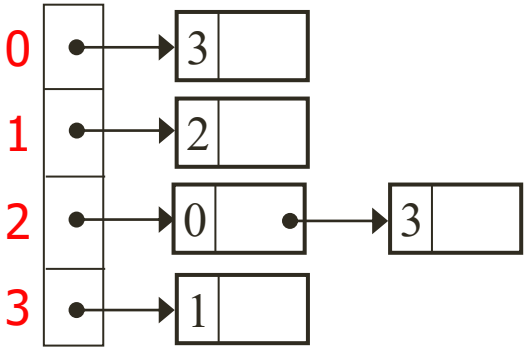
Ejercicios

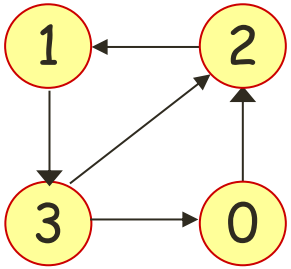
Ejercicio. Un grafo transpuesto *T* de un grafo *G* tiene el mismo conjunto de vértices pero con las direcciones de las aristas en sentido contrario, es decir, que una arista (u, v) en *G* se corresponde con una arista (v, u) en *T*.

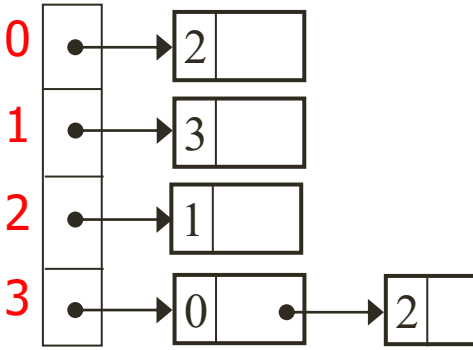
Diseña un método en la clase *GrafoDirigido* que permita obtener su grafo transpuesto:

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









```
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;
}
```