

Actividad 8. Uso TAD Grafo

Objetivo

Resolver problemas que hacen uso del TAD Grafo.

Procedimiento

- 1.- A partir de las transparencias disponibles en **Tema**, sección *Documentos e Ligazóns / Teoría/ TADGrafo.pdf*, comprender el TAD Grafo y su especificación.
- 2.- Resolver los ejercicios que se indican en esta actividad, utilizando el lenguaje java. Para probar su correcto funcionamiento se puede hacer uso del test disponible en Tema, sección Documentos e Ligazóns / Actividades / Test / **Actividad8Test.java**

Evaluación

Estos contenidos serán evaluados mediante una prueba que tendrá lugar el 15 de diciembre de 2017.

Tiempo estimado

4 horas

Ejercicios

1. Haciendo uso del TAD Grafo, escribe el método:

```
public static <E,T> Iterator<Vertice<E>> predecesores(Grafo <E,T> g, Vertice<E> v).
```


//Produce: devuelve un iterador sobre los predecesores del vértice *v* en el grafo *g*. Se dice que *w* es predecesor del vértice *v* si existe un arco que tenga por origen *w* y por destino *v*, es decir, el arco (*w,v*) pertenece al conjunto de arcos del grafo.
2. Escribe un método que dado un grafo y un vértice, determine si ese vértice es pozo o sumidero. Un vértice se dice **sumidero** si su grado de entrada es *n-1* y su grado de salida es 0. $|V| = n$, es decir, *n* es el número de vértices del grafo.
3. Escribe un método que devuelva cierto si un grafo es **regular**. Un grafo es regular si todos sus vértices tienen el mismo número de vértices adyacentes.
4. Escribe un método que dado un grafo dirigido y dos vértices devuelva cierto si existe un **camino** simple de un vértice a otro.
5. Se dice que un grafo dirigido es una **arborescencia** si existe un vértice, llamado raíz, desde el que se puede alcanzar cualquier otro vértice a través de un camino único. Implementa un método que determine si un grafo dirigido es una arborescencia. Caso de serlo, debe devolver la raíz.

Anexo:

```
public interface Grafo<E,T>{  
    public boolean esVacio();  
    public boolean estaVertice(Vertice<E> v);  
    public boolean estaArco(Arco<E,T> a);  
    public Iterator<Vertice<E>> vertices();  
    public Iterator<Arco<E,T>> arcos();  
    public Iterator<Vertice<E>> adyacentes (Vertice<E> v);  
    public void insertarVertice (Vertice<E> v);  
    public void insertarArco (Arco<E,T> a);  
    public void eliminarVertice (Vertice<E> v);  
    public void eliminarArco (Arco<E,T> a);  
}  
  
public class Vertice<E> {  
    private E etiqueta;  
    public Vertice(E o)  
    public E getEtiqueta()  
}  
  
public class Arco <E,T>{  
    private Vertice<E> origen, destino;  
    private T coste;  
    public Arco(Vertice<E> o, Vertice<E> d, T c)  
    public Vertice<E>E getOrigen()  
    public Vertice<E> getDestino()  
    public T getEtiqueta()  
}
```