# Estructuras de Datos. Febrero 2016

Un **ciclo euleriano** en un grafo es un camino que empieza y termina en el mismo vértice y que visita todas las aristas del grafo exactamente una vez. Leonhard Euler conjeturó en 1736 que un grafo conectado tiene ciclo euleriano si y solo si todos sus vértices tienen grado par. Años más tarde Carl Hierholzer demostró esta conjetura. El objetivo del examen es escribir programas para calcular un ciclo euleriano de un grafo euleriano usando un algoritmo propuesto por el propio Hierholzer. En el examen asumiremos que los grafos son conexos y tienen al menos dos vértices.

## Haskell

Descarga del campus virtual el archivo comprimido que contiene las fuentes para resolver el problema y comprobar tu solución. Completa las definiciones de funciones del fichero `DataStructures\Graph\EulerianCycle.hs`. Este es el único fichero que tienes que modificar y el único fichero que debes subir a través del enlace de entrega del campus virtual. Ten en cuenta que tu solución debe compilar sin errores para que se considere adecuada.

**H.1)** (0.5 puntos) Un grafo es euleriano si tiene ciclo euleriano. Basándote en la conjetura de Euler, define una función `isEulerian` que recibe un grafo y devuelve `True` si y solo si es euleriano.

**H.2)** (0.5 puntos) La biblioteca `DataStructures.Graph.Graph` exporta la función:

```
deleteVertex :: (Eq a) => Graph a -> a -> Graph a
```

para borrar un vértice de un grafo, y la función:

```
deleteEdge :: (Eq a) => Graph a -> (a,a) -> Graph a
```

para borrar una arista de un grafo.
Un vértice se dice **aislado** si su grado es cero. Define una función `remove` que toma un grafo y una arista y borra la arista del grafo, así como todos los nodos que queden aislados después de borrar la arista.

**H.3)** (0.75 puntos) Sean `g` un grafo euleriano y `v0` uno de sus vértices (puedes asumir estas precondiciones en tu programa). Define una función `extractCycle` que tome el grafo `g` y el vértice `v0` y extraiga un ciclo (no necesariamente euleriano) de `g` que comienza en `v0`. Utiliza para ello el siguiente algoritmo voraz:

> Inicialmente el ciclo contiene solo el vértice `v0`. Sea `v` inicialmente el vértice `v0`, y `u` algún sucesor del vértice `v`, añade `u` al ciclo, elimina la arista (v,u) así como cualquier vértice aislado del grafo g (utiliza la función del apartado anterior). Repite estos pasos desde el vértice `u` hasta alcanzar el vértice inicial `v0`.

Además del ciclo extraído, la función `extractCycle` debe devolver el nuevo grafo que queda tras borrar todas las aristas contenidas en el ciclo extraído, así como todos los vértices que han quedado aislados tras la extracción.

**H.4)** (0.75 puntos) Sean `xs` un ciclo euleriano parcial[1] e `ys` un ciclo no vacío. La función `connectCycles` devuelve una combinación de ambos ciclos. Si `xs` es vacío, entonces el resultado es `ys`. Si `xs` no es vacío, podemos asumir la siguiente precondición: el primer vértice de `ys`, al que llamaremos `y`, aparece en `xs`. En este caso, el ciclo combinado resultante se obtiene reemplazando la primera aparición del vértice `y` en `xs` por `ys`. Por ejemplo:

```
connectCycles [] [A,B,C,D,E,A]         => [A,B,C,D,E,A]
connectCycles [A,B,C,D,E,A] [C,F,G,C] => [A,B,C,F,G,C,D,E,A]
```

Define la función `connectCycles`.

[1] Un ciclo euleriano que no está completo; se usa en el apartado **H.6**, que va extendiendo un ciclo euleriano parcial hasta que esté completo.

**H.5)** (0.5 puntos) Define una función `vertexInCommon` que toma como parámetros un grafo `g` y un ciclo `xs` y devuelve un vértice que aparece tanto en el grafo `g` como en el ciclo `xs`. Puedes asumir que siempre hay al menos un vértice en común.

**H.6)** (0.75 puntos) Define una función `eulerianCycle` que toma un grafo y devuelve un ciclo euleriano del grafo. Si el grafo no es euleriano, se debe terminar con error. En otro caso, se deben extraer y conectar ciclos del grafo hasta que éste quede vacío. La combinación de todos los ciclos extraídos será un ciclo euleriano. Empieza extrayendo un ciclo desde un vértice cualquiera del grafo. Éste es el primer ciclo euleriano parcial. Mientras el grafo resultante de la extracción no esté vacío, repite el siguiente proceso: toma un vértice del grafo resultante que también aparezca en el ciclo euleriano parcial, extrae un ciclo del grafo que comience en ese vértice, y conecta ese nuevo ciclo al ciclo euleriano parcial para obtener el nuevo ciclo euleriano parcial. Al finalizar el proceso, el ciclo euleriano estará completo. Utiliza las funciones definidas anteriormente para implementar el algoritmo.

## Java

Descarga del campus virtual el archivo comprimido que contiene el proyecto Eclipse para resolver el problema y comprobar tu solución. Completa las definiciones de métodos del fichero `dataStructures\graph\EulerianCycle.java`. Este es el único fichero que tienes que modificar y el único fichero que debes subir a través del enlace de entrega del campus virtual. Ten en cuenta que tu solución debe compilar sin errores para que se considere adecuada.

**J.1)** (0.5 puntos) Define un método:

```
private static <V> boolean isEulerian(Graph<V> g)
```

que devuelva `true` si y solo si el grafo es euleriano.

**J.2)** (0.5 puntos) Define un método:

```
private static <V> void remove(Graph<V> g, V v, V u)
```

similar a la función Haskell `remove` del apartado **H.2**.

**J.3)** (0.75 puntos) Define un método:

```
private static <V> List<V> extractCycle(Graph<V> g, V v0)
```

similar a la función Haskell `extractCycle` del apartado **H.3**. En este caso, en lugar de devolver una tupla con el ciclo y el grafo, el ciclo extraído se devuelve como resultado del método, y el grafo `g` pasado como parámetro se modifica durante la extracción.

**J.4)** (0.75 puntos) Define un método:

```
private static <V> void connectCycles(List<V> xs, List<V> ys)
```

similar a la función Haskell `connectCycles` del apartado **H.4**. En este caso, el método es `void` y el resultado queda almacenado en el primer parámetro `xs`.

**J.5)** (0.5 puntos) Define un método:

```
private static <V> V vertexInCommon(Graph<V> g, List<V> xs)
```

similar a la función Haskell `vertexInCommon` del apartado **H.5**.

**J.6)** (0.75 puntos) Define un método:

```
private static <V> List<V> eulerianCycle(Graph<V> g)
```

similar a la función Haskell `eulerianCycle` del apartado **H.6**.