PROGRAMACIÓN DECLARATIVA
PRÁCTICA FINAL Curso 2019/2020

Objetivo de la práctica: Escribir un programa en Haskell que implemente una serie de funciones básicas sobre grafos dirigidos. Para hacer esta práctica te vendrá bien repasar los conceptos fundamentales de la teroía de grafos.

Descripción del problema: Un grafo dirigido G consta de un conjunto de vértices V y un conjunto de arcos orientados A, que son pares cuyas componentes están en V. Los vértices de los grafos que consideramos se denotan con alguna de las letras A, B, C, D, E, F. Los siguientes son tres ejemplos de grafos:

$$\begin{array}{lcl} G_1 & \equiv & (\{B,D,E,C\},\{(D,E),(E,B),(C,B),(E,C)\}) \\ G_2 & \equiv & (\{D,F,E\},\{(D,F),(E,D),(D,E),(F,E)\}) \\ G_3 & \equiv & (\{A,C,D\},\{(A,C),(C,D),(A,D)\}) \end{array}$$

Consideraciones:

- Para representar en Haskell nuestros grafos partimos de las siguientes definiciones de tipos de datos:

  ```
  data Vertice = A|B|C|D|E|F
  data Grafo = G [Vertice] [(Vertice,Vertice)]
  ```

- El tipo `Grafo` debe ser instancia de las clases `Read` y `Show`.

- Deben existir las siguientes funciones Haskell:

  `es_grafo g = True` si la expresión `g` (que debe ser de tipo `Grafo`) representa realmente un grafo, es decir el conjunto de vértices de `g` no es vacío y no tiene vértices repetidos, y los elementos del conjunto de arcos de `g` son realmente arcos dentro del conjunto de vértices de `g`. Vale `False` en caso contrario.

  `mat_ady g` = matriz de adyacencia del grafo `g`.

  `grados_pos g` = lista de los grados positivos de los vértices del grafo `g`.

  `grados_neg g` = lista de los grados negativos de los vértices del grafo `g`.

  `camino_lng g v n` = lista de los caminos en el grafo `g` con origen en el vértice `v` y de longitud `n`.

  `conexo g = True` si `g` es un grafo conexo; `False` en caso contrario.

- El tipo `Grafo` debe ser instancia de la clase `Eq`. Se considera que dos grafos son iguales si son *isomorfos*.

Primera parte

Declarar los tipos anteriores y programar las funciones definidas anteriormente, teniendo en cuenta las consideraciones previamente enunciadas y las indicaciones que siguen.

- Hay que declarar los tipos de todas las funciones que se programen, incluidas las funciones auxiliares que pudieran necesitarse.
- Se pueden usar todas las funciones de `Prelude`, es decir, las que se cargan con el sistema, pero no se puede importar ningún otro módulo, lo que quiere decir que, por ejemplo, si se usan operaciones con listas que no están en `Prelude` hay que programarlas.
- Para poder realizar ejemplos y evaluar la práctica, deben incluirse al menos seis grafos concretos (`g1,g2,g3,g4,g5,g6`), definidos mediante constantes de tipo `Grafo`. Los tres primeros, `g1,g2,g3` deben corresponder a los grafos G_1, G_2, G_3 de arriba, y al menos dos de estos seis deben ser isomorfos entre sí y también tiene que haber por lo menos dos no isomorfos.

Segunda parte

Implementa las siguientes acciones de E/S, declarando tipos adecuados:

- `leegrafo` pide al usuario que introduzca los vértices y los arcos de un grafo uno a uno y comprueba que los datos introducidos corresponden efectivamente a un grafo. Pide rectificación en caso contrario.
- `muestra_matriz` invoca a la función `leegrafo` y muestra su matriz de adyacencia en forma de matriz.
- `muestra_caminos` invoca a la función `leegrafo` y comprueba si el grafo introducido es conexo. En caso afirmativo muestra, para un vértice v del grafo, un camino existente desde v al resto de los vértices del grafo, lo que permite afirmar que el grafo es conexo. Por ejemplo, para el grafo G_3, una posibilidad, entre otras, sería mostrar estos dos caminos `A-C` y `A-C-D`. Si el grafo no es conexo, escribe un mensaje indicándolo.

Entrega de la práctica y calificación:

- La entrega se realizará a través del Campus Virtual y consistirá en un solo fichero .hs no comprimido, en el que las explicaciones irán como comentarios Haskell. El nombre del autor debe aparecer como comentario en la primera línea.
- Fecha límite para la entrega: **17 de enero de 2020**.
- La nota de la práctica supone el **10 % de la nota final (1 punto)**.
- Para obtener 0,7 puntos basta dar una solución razonable y bien explicada a la primera parte. La eficiencia no es nuestra mayor preocupación, pero se valorará muy positivamente la utilización con acierto de funciones de orden superior y listas intensionales. Para obtener los 0,3 puntos restantes hay que programar la segunda parte.

El trabajo es INDIVIDUAL. La copia de otros compañeros o de cualquier otra fuente, así como facilitar la copia a otros, será severamente castigado en la calificación **global** de la asignatura.