Facultad de Ciencias Exactas, Ingeniería y Agrimensura
Escuela de Ciencias Exactas y Naturales
Departamento de Ciencia de la Computación
Estructura de Datos y Algoritmos II

Nombre y Apellido: Legajo:

Examen Final

1. Dado el siguiente TAD para grafos nodirigidos.

a) Nombre de tipo: Graph
Usa: Bool, Set

b) Operaciones:

tad *Graph* $(V : Set)$

null	: *Graph* V	
addVertex	: *Graph* $V \to V \to$ *Graph* V	-- agrega un vértice
addEdge	: *Graph* $V \to (V, V) \to$ *Graph* V	-- agrega una arista
delVertex	: *Graph* $V \to V \to$ *Graph* V	-- elimina un vértice y las aristas incidentes en éste
delEdge	: *Graph* $V \to (V, V) \to$ *Graph* V	-- elimina una arista
adyacents	: *Graph* $V \to V \to V \to$ *Bool*	-- determina si dos vértices son adyacentes
neightboors	: *Graph* $V \to V \to$ *Set* V	-- calcula el conjunto de vecinos de un vértice
isPath	: *Graph* $V \to V \to V \to$ *Bool*	-- determina si hay un camino entre los 2 vértices

c) Especificación algebraica.

$$
\begin{aligned}
&\textit{delEdge null } (v, w) &&= \textit{null}\\
&\textit{delEdge } (\textit{addVertex } g\ x)\ (v, w) &&= \textit{delEdge } g\ (v, w)\\
&\textit{delEdge } (\textit{addEdge } g\ (v, w))\ (x, y) &&= \textbf{if } ((x \equiv v) \wedge (y \equiv w) \vee (x \equiv w) \wedge (y \equiv v)) \textbf{ then } g\\
&&&\quad \textbf{else } \textit{addEdge } (\textit{delEdge } g\ (x, y))\ (v, w)
\end{aligned}
$$

...

i) Completar la especificación algebraica del TAD con la especificación de las operaciones *neightboors* y *path*.

ii) Una forma de representar un grafo es mediante su matriz de adyacencias. Para ello se crea una matriz cuadrada cuyas filas y columnas representan nodos del grafo y el valor de la posición (i,j) en matriz es igual a 1 si existe una arista entre el vértice representado por i y el vértice representado por j. Esta representación permite definir las operaciones *addEdge*, *delEdge*, *adyacents* y *neighbours* de forma eficiente, con costo de orden $O(V)$, siendo V el conjunto de vértices del grafo.

Suponiendo que los grafos están representados por su matriz de adyacencias y que éstos se implementan en Haskell mediante el siguiente tipo:

```
data Bit = Zero | One deriving Eq
type Vertex = Int   -- representado por su posición en la matriz
type Graph = [[Bit]]
```

Definir las funciones *neighbours* y *delEdge* del TAD.

iii) Bajo el mismo supuesto, dar una implementación de $inverso :: Graph \to Graph$ como una lista por comprensión (sin utilizar funciones auxiliares). El inverso de un grafo g es un grafo g' con el mismo conjunto de vértices y tal que dos vértices de g' son adyacentes si y sólo si no son adyacentes en g. Notar que el inverso de un grafo g puede contener bucles si éstos no están en g.

2. Debido a las bajas en la venta de celulares, una empresa decide hacer un remate de los mismos por internet. Las personas tendrán dos horas para hacer una única oferta sobre el precio de los mismos, pero sólo podrán terminar su compra aquellas ofertas que superen al promedio de las ofertas anteriores y superen cierto valor X.

Definir una función `ventas` : $Seq\ Float \to (Seq\ Float, Float)$ que dada una secuencia cronológica con los precios ofertados, devuelva una secuencia con los precios correspondientes a personas que pudieron terminar la compra junto con la menor oferta terminada en compra o -1 si ninguna oferta terminó en compra.

Definir ventas utilizando la función *scan*, con profundidad de orden $O(lg(n))$, donde n es el largo de la secuencia. Calcular costos.

3. Suponiendo una representación de cadenas de caracteres con árboles binarios, definidos con el siguiente tipo de datos:

data *Cadena* = $E \mid L\ Char \mid N\ Cadena\ Cadena$

donde el recorrido inorder del árbol da el orden de los caracteres de la cadena, definir en Haskell, de manera eficiente, paralelizando cuando sea posible, las siguientes funciones:

a) La función $palidromo :: Cadena \to Bool$, que determina si una cadena es palíndromo (es decir, es una palabra que es igual si se lee de izquierda a derecha que de derecha a izquierda).

b) La función $crear :: Int \to (Int \to Char) \to Cadena$, que dado un número n positivo y una función f, cree una cadena de longitud $n+2$ delimitada por el caracter '*', donde cada caracter que no sea primero ni último sea el resultado de aplicar f sobre la posición del caracter en la cadena. Por ejemplo,

$$crear\ 5\ f = \langle\ f\ 1, f\ 2, f\ 3, f\ 4, f\ 5\ \rangle$$