Examen de Programación 2
15 de diciembre de 2016

Generalidades:

a) La prueba es individual y sin material.
b) La duración es **3**hs.
c) Sólo se contestan dudas acerca de la letra.
d) Escriba las hojas de un solo lado y con letra clara.
e) Numere cada hoja, indicando en la primera el total.
f) Escriba número de cédula y nombre en cada hoja.

Ejercicio 1 (50 puntos)

Se quiere determinar cuáles son los productos más comprados en un supermercado. Para esto se cuenta con un TAD `Compra` que representa los pares *(codigo, cantidad)*, donde código representa un código alfanumérico de producto y cantidad es la cantidad de veces que dicho producto fue comprado. En particular, en el TAD `Compra`, se definen las operaciones selectoras `obtenerCodigo` y `obtenerCantidad`. Además se cuenta con el TAD `ListaCompra`, con las operaciones usuales del TAD Lista, para modelar la lista de compras. *Recuerde que la inserción se hace al comienzo.*

Se quiere implementar una función `ListarKMasComprados`, que dada una lista de productos comprados representada con el TAD `ListaCompra` y un entero **k**, devuelve otra `ListaCompra` con los **k** pares *(codigo, cantidad)* que corresponden a los k productos más comprados. Estos pares se devuelven ordenados en forma decreciente según cantidad de compras. No hay un criterio de desempate preestablecido en caso de empates en la cantidad de compras, quedando esto librado a la implementación. Si **k** es mayor que el largo de la lista se devuelven todas las compras. El encabezado de la función `ListarKMasComprados` es el siguiente:

```
ListaCompra ListarKMasComprados (ListaCompra lst; unsigned int k);
```

A continuación se presentan algunos ejemplos de ejecuciones.

```
lst =[(GALL03,3),(POLLO99,2),(PAN35,3),(LECH07,5),(YOGU55,3)]

ListarKMasComprados(lst,0)= []
ListarKMasComprados(lst,1)= [(LECH07,5)]
ListarKMasComprados(lst,2)= [(LECH07,5),(YOGU55,3)] o
                            [(LECH07,5),(GALL03,3)] o
                            [(LECH07,5),(PAN35,3)]
```

(a) (15 puntos) Especifique el TAD `PrioridadCompra` como una Cola de Prioridad acotada de elementos del tipo `Compra`, donde un elemento tiene prioridad sobre otro si el valor de su campo **cantidad** es menor. El tamaño de la Cola de Prioridad se define al momento de crearla. Incluya pre y post condiciones.

(b) (20 puntos) Implemente `ListarKMasComprados` usando una estructura auxiliar de tipo `PrioridadCompra` de tamaño k, las operaciones especificadas en (a) y las de los TADs `Compra` y `ListaCompra`. La lista original **lst** se puede recorrer una sola vez.

(c) (15 puntos) Suponiendo que **k** es acotado por una constante **MAX_K** , explique qué implementación del TAD `PrioridadCompra` de la parte (a) permite realizar `ListarKMasComprados` en O(n * log k), donde n es la cantidad de productos. Describa propiedades de la implementación propuesta y tiempos de ejecución en el peor caso para cada una de las operaciones especificadas.

Ejercicio 2 (50 puntos)

(a) Considere la siguiente implementación del tipo `Arbol`, de árboles binarios de un tipo genérico `T`, donde el árbol vacío se implementa con `NULL`.

```
struct nodo {
  T dato;
  nodo *izq, *der;
};
typedef nodo *Arbol;
```

I. (10 puntos) Implemente la función `int Altura(Arbol t)`, que devuelve la altura del árbol `t`. La altura del árbol vacío es 0. A modo de ejemplo considere el árbol de la Figura 1, en el cual una llamada a la función `Altura` sobre el árbol que tiene raíz en `F` devuelve 1, mientras que para el que tiene raíz en `D` devuelve 3.

II. (15 puntos) Implemente la función `bool EsNodoBalanceado(Arbol t)`, que devuelve `true` si y sólo si la raíz de `t` cumple la condición de balanceo de los árboles `AVL`. Se considera que el árbol vacío está balanceado. A modo de ejemplo, una llamada a la función `EsNodoBalanceado` sobre los arboles de la Figura 1 que tienen como raíz a `A` y a `E` debe devolver `false`.

(b) (25 puntos) Considere ahora la siguiente implementación del tipo `Arbol`, donde se agrega un atributo `es_arbol_balanceado` que es `true` si y sólo si el nodo y cada uno de sus descendientes cumplen la condición de balanceo de los árboles `AVL`.

```
struct nodo {
  T dato;
  bool es_arbol_balanceado;
  nodo *izq, *der;
};
typedef nodo *Arbol;
```

Implemente el procedimiento `EstablecerBalanceo(Arbol & t)`, que establece el valor correcto del atributo `es_arbol_balanceado` para cada nodo de `t`. **Su solución puede visitar cada nodo sólo una vez.** A modo de ejemplo, en el árbol de la Figura 1, el atributo `es_arbol_balanceado` debe ser `false` en los nodos `A`, `B` y `E`.

Observaciones:

- La implementación de todas las funciones se debe hacer accediendo a la representación del tipo `Arbol`.
- En ambas partes se puede asumir que están definidas las funciones `max` (que devuelve el máximo de dos enteros) y `abs` (que devuelve el valor absoluto de un entero).

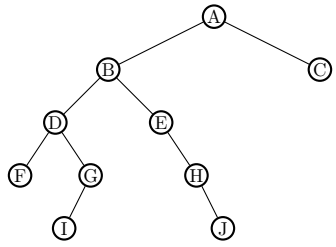

Figura 1: Ejemplo para el Ejercicio 2